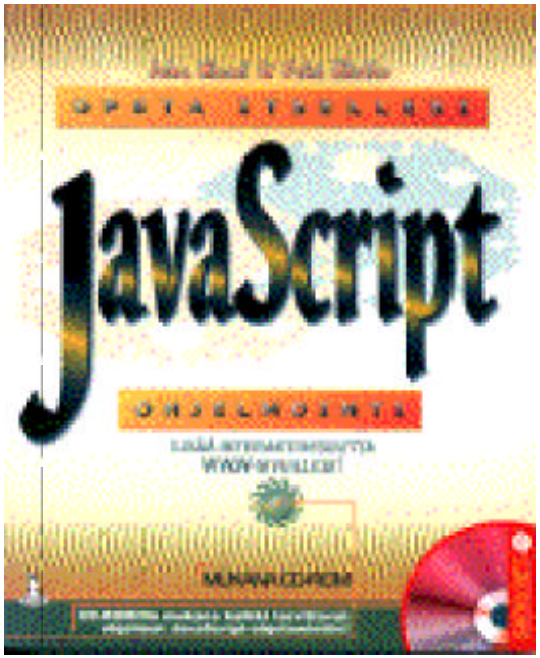


Suomen
Atk-kustannus Oy
1997

Juha Niemi
Petri Kiuttu

Opeta itsellesi JavaScript -ohjelmointi

Opeta itsellesi JavaScript-ohjelmointi

Kaikki oikeudet pidätetään. Tämän julkaisun jäljentäminen ilman tekijän kirjallista lupaa painamalla, monistamalla, äänittämällä tai muulla tavoin on tekijänoikeuslain mukaisesti kielletty.

Copyright

Kirjoittajat	Petri Kiuttu Juha Niemi
Kustantaja	Suomen ATK-kustannus Oy PL 68 02601 ESPOO Puhelin (09) 4765 8550 Telekopio (09) 4765 8595
Kansi	Frank Schaumont
Taitto	Juha Niemi
ISBN	951-762-447-6
Sähköinen laitos	Suomen ATK-kustannus Oy Helsinki 1997

Sisällysluettelo

1. YLEISTÄ 9
1.1 HTML 9
1.2 JavaScript-ohjelmointi 9
1.3 CGI-ohjelmat 10
1.4 Java-maailma 10
1.5 Mitä ovat Java ja JavaScript? 11
1.6 JavaScriptiä tukevat selaimet 11
1.7 Netscape Navigator 3.0 12
1.8 JavaScriptin mahdollisuudet 14
1.9 JavaScript-kieli 14
1.10 Internet-osoitteita 15

2. HTML-KIELI 19
2.1 HTML-kielen perusteet 19
2.2 HTML-komennot 21
2.2.1 Ensimmäinen ja viimeinen komento 21
2.2.2 Otsikkotiedot 21
2.2.3 Dokumentin nimi 21
2.2.4 Dokumentin osoite 22
2.2.5 Dokumentin runko 22
2.2.6 Omat Otsikkotiedot 23
2.2.7 Ääkkoset (skandit) ja erikoismerkit 24
2.2.8 Fontit 24
2.2.9 Rivinvaihto 25
2.2.10 Kappale 25
2.2.11 Otsikko 26
2.2.12 Väliviiva 26
2.2.13 Lihavointi 26
2.2.14 Kursiivi 26
2.2.15 Alleviivaus 26
2.2.16 Kommentti 27
2.2.17 Esimuotoiltu Teksti 27

2.2.18 Numeroitu lista 28
2.2.19 Ranskalaiset viivat 30
2.2.20 Määritelmälistat 31
2.2.21 Sisennetyt listat 33
2.2.22 Keskitys 34
2.2.23 Rivinvaihdon estäminen 34
2.2.24 Taulukko 35
2.2.25 Linkit 38
2.2.26 Indeksitiedosto 39
2.2.27 Automaattisesti päivittyvä dokumentti 39
2.2.28 Kuvat 40
2.2.29 Kehykset (Frames) 43

2.3 Interaktiiviset kuvat 46
2.3.1 Interaktiivinen kuva dokumentissa 48

2.4 Protokollat 49
2.4.1 Http 49
2.4.1 49
2.4.2 Ftp 49
2.4.2 49
2.4.3 News 50
2.4.3 50
2.4.4 Telnet 50
2.4.5 Mailto 51
2.4.5 51
2.4.6 Wais 51
2.4.6 51
2.4.7 File 51
2.4.7 51

2.5 Mime-tyypit 52

2.6 Lomakkeet (Forms) 53
2.6.1 Tekstikentät 54
2.6.2 Tekstialue 55
2.6.3 Salasanakenttä 57
2.6.4 Piilotetut kentät 58
2.6.5 Painikkeet 59
2.6.6 Valikot 61
2.6.7 Lähetyspainikkeet 63

2.7 SSI-komennot 65
2.7.1 Tiedoston lisäys 65
2.7.2 Muuttujan tulostaminen 66
2.7.3 Muiden dokumettien tiedot 66

2.7.4 Komentojen ja ohjelmien tulosteet 67
2.7.5 Muotoilukäskyt 67

3. JAVASCRIPT-KIELEN PERUSTEET 69

3.1 JavaScript ja Java 69

3.2 Kielen rakenne ja kirjoitusasu 70

3.3 Kommentit 71

3.4 Muuttujat ja tietotyypit 72

3.5 Merkkijonot 77

3.6 Operaattorit 78

3.7 Kontrollilauseet 83

3.8 JavaScript-kielen olioista 86

3.9 Funktiot ja metodit 89
3.9.1 This-osoitin 94

3.10 Sisään rakennetut funktiot 95

4. OHJELMOINTI JAVASCRIPT-KIELELLÄ 101

4.1 Tapahtumafunktiot 102

4.2 Olioiden luonti 104

4.3 Metodin luominen 106

4.4 Taulukkomuuttujan luominen 109

4.5 Math-olio 112

4.6 String-olio 128

4.7 Date-olio 138

4.8 Form-olio 148

4.9 Form-olion elementit 155
4.9.1 Button-olio 156
4.9.2 Checkbox-olio 159
4.9.3 Hidden-olio 161
4.9.4 Link-olio 161
4.9.5 Password-olio 162

4.9.6 Radiobutton-olio 163
4.9.7 Reset-olio 167
4.9.8 Selection-olio 169
4.9.9 Submit-olio 175
4.9.10 Text-olio 176
4.9.11 Textarea 180

4.10 Frame-olio 180

4.11 Navigaattori-olio 183

4.12 Yleiset Navigaattori-oliot 185
4.12.1 Document-olio 185
4.12.2 History-olio 192
4.12.3 Location-olio 194
4.12.4 Window-olio 195

4.13 Cookie-mekanismi 218
4.13.1 Cookie:n toimeenpano CGI-ohjelmalla 220
4.13.2 Cookie:n toimeenpano JavaScriptillä 221

5. CGI-OHJELMAT 223

5.1 Yleistä 223

5.2 CGI-ohjelma unix-komentotiedostoilla 225
5.2.1 Ensimmäinen komentotiedosto 227
5.2.2 Käyttöjärjestelmäkomennot ja muuttujat komentotiedostossa 230
5.2.3 Komentotiedostojen parametrit 239
5.2.4 Parametrien välitys CGI-ohjelmaan URL:n avulla 239

5.3 CGI-ohjelmien ympäristömuuttujat 240
5.3.1 Javascript komentotiedostoissa 244

5.4 Lomakkeiden käsittely 246
5.4.1 Lähetystapa GET 247
5.4.2 Lähetystapa POST 255

HAKEMISTO 291

Johdanto

Tämä opas on tarkoitettu juuri sinulle, joka olet kiinnostunut Internetistä ja omien kotisivujen ylläpidosta ja kehittämisestä sekä suuremmista sivukokonaisuuksista. HTML- ja JavaScript-kielen sekä CGI-ohjelmien avulla opit nopeasti tekemään näyttäviä interaktiivisia ja/tai multimediadokumentteja WWW-palvelimellesi. Parhaaseen tulokseen pääset yhdistämällä HTML:n, JavaScriptin ja CGI-ohjelmat. Tässä kirjassa on eväät siihen!

Kirja kertoo JavaScript- ja CGI-ohjelmoinnista kaiken oleellisen, jonka tarvitset luodaksesi toimivia www-sivukokonaisuuksia. Et tarvitse ennestään JavaScriptin osalta kokemusta ohjelmoinnista, etkä tuntemusta käyttöjärjestelmistä. Riittää, kun olet tutustunut WWW:hen, esimerkiksi Netscapella. Kirjassa käydään lävitse HTML-kieli, joten sen tunteminenkaan ei ole välttämätöntä, mutta suotavaa.

Kirjan kohderyhmä

JavaScript- ja HTML-kielen sekä CGI-ohjelmien avulla opit nopeasti tekemään näyttäviä interaktiivisia ja multimediaesityksiä WWW-sivuillesi.

Kirjasta on ensisijaisesti hyötyä

- niille, jotka haluavat tutustua WWW:n syvällisempään ohjelmointiin
- niille, jotka haluavat ymmärtää CGI-ohjelmointia ja haluavat luoda välitöntä interaktiivisuutta
- niille, jotka ovat opiskelleet jotakin ohjelmointikieltä, esimerkiksi C-kieltä.

Kirjassa on lukuisia esimerkkejä ja kokonaisten sivujen lähdekoodeja, joita voit käyttää elävöittämään omia sivujasi. Kirjan mukana tulevalla CD-ROM-levyllä on valmiita WWW-sivuja sekä tarpeellisia vapaasti levitettäviä varusohjelmia.

Porissa 1. lokakuuta 1996

Petri Kiuttu
petrik@evmech.fi

Juha Niemi
jniemi@it.spt.fi

1. YLEISTÄ

Tämä kirja jakaantuu kolmeen eri osaan. Ensimmäinen osa käsittelee HTML-kieltä ja toinen osa JavaScript-ohjelmointia ja kolmas CGI-ohjelmia. Kaikille osille on luku 1 yhteinen, jossa on esittelyä selaimista, HTML-kielestä ja JavaScript-kielestä.

Luku 1	Esittely
Luku 2	Esittelee Java-maailmaa

1.1 HTML

HTML-kieleen tutustuminen aloitetaan luvusta kolme. Luvussa käsitellään koko HTML-kieli siinä laajuudessa kuin sitä tämän kirjan muissa osissa tarvitaan. Luku sisältää myös runsaasti selventäviä esimerkkejä. Kirjan HTML-oppimäärä riittää myös sellaisenaan www-dokumenttien tekoon.

Luku 3	HTML

1.2 JavaScript-ohjelmointi

JavaScript-ohjelmointi alkaa luvusta neljä. Alussa tutustutaan JavaScriptin kielioppiin, sekä pienien esimerkkien avulla JavaScript-kielen käyttöön HTML-sivuilla. Seuraavassa luvussa aloitetaan varsinainen JavaScript-ohjelmien teko. Viimeisessä luvussa (6) Käsitellään CGI-ohjelmia.

Luku 4	JavaScript
Luku 5	Ohjelmointi JavaScriptillä

Javan ja JavaScriptin käyttö vaatii vähintään Netscape 2.0 -selaimen. Nykyään myös muihin selaimiin on lisätty Java-tukea (esimerkiksi Win-95:n Explorer). Javaa käytettäessä selaimen sekä käyttöjärjestelmän on oltava 32-bittisiä. Käytännössä se tarkoittaa Windows 95 tai NT:n käyttöä.

Koska WWW ja kaikki siihen liittyvä kehittyy nopeasti, on todennäköistä, että tämänkin kirjan käsittelemiin aiheisiin tulee lisäyksiä ja mahdollisesti jopa muutoksia. Perusteet eivät kuitenkaan muutu miksikään. Kirjan suomennoksissa on pyritty käyttämään vakiintuneita termejä, jos sellaisia on ollut olemassa.

1.3 CGI-ohjelmat

CGI-ohjelmia käsitellään luvussa kuusi. Luvussa käsitellään CGI-ohjelmoinnin periaatteita sekä varsinaista ohjelman tekemistä komentotiedostoilla (unix-ympäristö) ja C- sekä C++ -kielellä (voidaan kääntää kaikkiin ympäristöihin). Kirjassa ei käsitellä C-ohjelmointia opetusmielessä.

Luku 6 CGI-ohjelmat

1.4 Java-maailma

Tässä luvussa tutustutaan Netscape 3.0:n ominaisuuksiin, sekä kerrotaan JavaScriptistä. Tämä luku toimii siis pohjana kirjan myöhemmille luvuille.

1.5 Mitä ovat Java ja JavaScript?

Java on verkko-ohjelmointikieli. Java-kielellä voi toteuttaa monenlaisia ohjelmia, joita voi ajaa Netscapessa. Java-kieltä voidaan verrata C++-kieleen, sillä se on ulkoasultaan samanlaista. Javan oppiminen ja sen täysipainoinen ymmärtäminen on tietenkin vaikeampaa HTML-kieleen verrattuna. Javan ominaisuuksia ovat muun muassa:

- oliopohjaisuus
- yksinkertaisuus
- siirrettävyys
- tehokkuus
- dynaamisuus
- turvallisuus
- monisäikeisyys
- jaettavuus

Netscape Communications ja Sun Microsystems ovat kehittäneet myös helpomman tavan ohjelmoida sovelluksia, lähinnä interaktiivisia sovelluksia. Synnytettiin kieli nimeltä JavaScript. JavaScript on siten myös verkko-ohjelmointikieli. JavaScript ei vastaa laajuudeltaan Javaa, mutta sillä on helpompi toteuttaa ohjelmia, koska HTML-kieltä voidaan hyödyntää ohjelmissa. JavaScript tulee luultavasti syrjäyttämään Javan koko kansan WWW-sivujen ohjelmointikielenä.

1.6 JavaScriptiä tukevat selaimet

Tällä hetkellä JavaScriptiä tukevat jo useat selaimet. Yleisin ja suosituin selain tietenkin on Netscape Navigator 3. Spyglassin Mosaic-selain ja Oracle PowerBrowser tukevat myös JavaScriptiä.

- Netscape **Navigator 2.0** ja uudemmat versiot
- Microsoft **Internet Explorer 3.0**

- Spyglass **Mosaic**
- Oracle **PowerBrowser**

Netscape on selaimista ainoa kaupallinen. Netscapen ilmaisen 30 päivää toimivan demon voi imuroida verkosta (http://www.netscape.com). Muilla selaimilla Internetissä olevia Java-ohjelmia voi katsella ilmaiseksi.

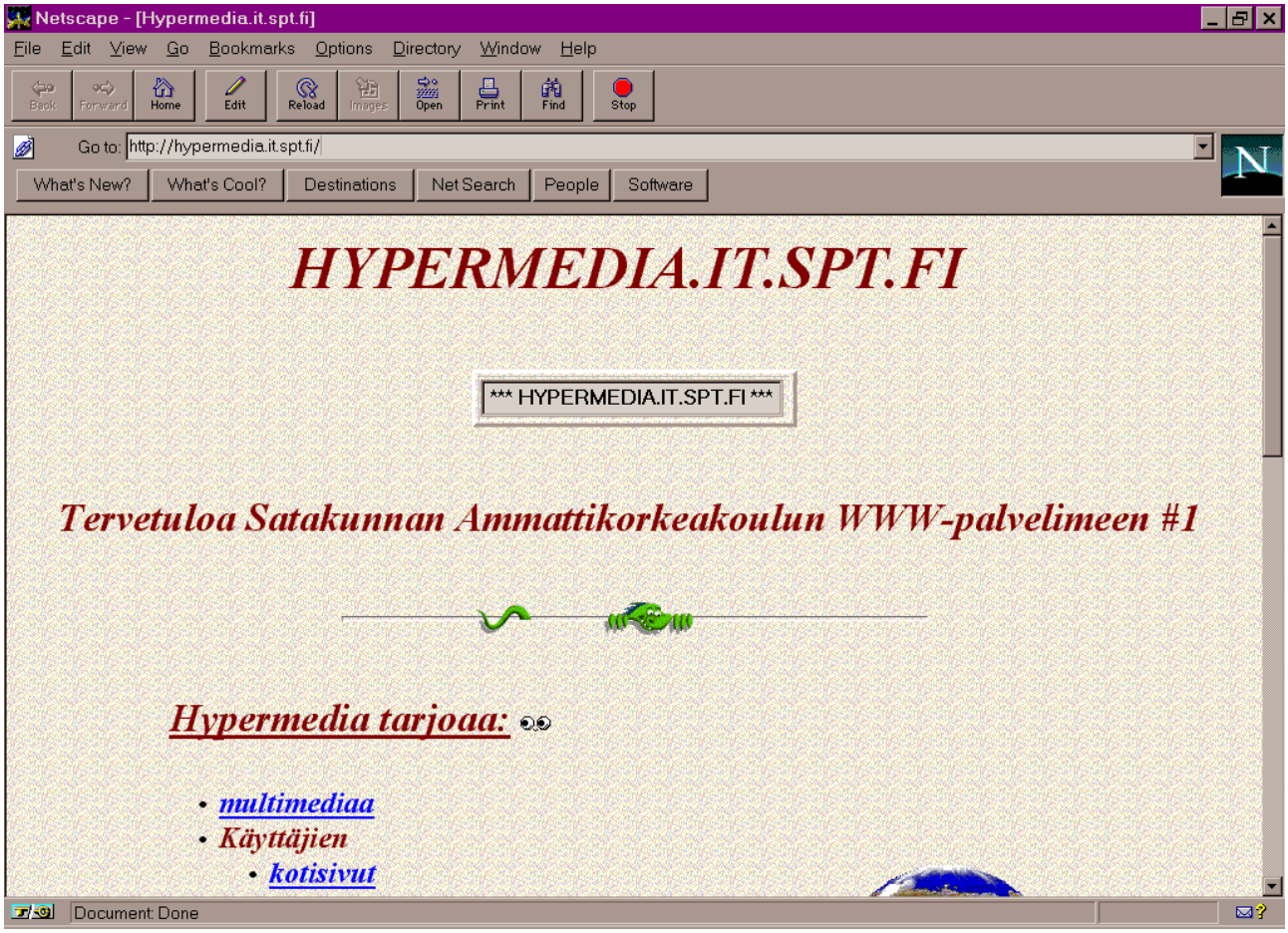
1.7 Netscape Navigator 3.0

Netscape on vakiinnuttanut paikkansa WWW-selaimien ykköseksi, eikä suotta sillä se on ehdottomasti käyttäjäystävällisin, ja erittäin toimiva kokonaisuus.

Netscapesta on saatavissa monta eri versiota. Navigator 1.0 on monille tuttu perusselain ja vieläpä ilmaisohjelma. Se on kuitenkin nykyään liian vanha, koska se ei tue nykypäivän kaikkia ominaisuuksia. Tänä päivänä ehdoton ykkönen onkin Netscape Navigator 3.0. Tämä selain tukee HTML 1, HTML 2, HTML 3 standardeja, sekä Netscapen omia laajennuksia. Tämän kirjan kannalta tärkein ominaisuus on parannettu JavaScript tuki. Lisäksi **Gold**-versiossa on sisäänrakennettu HTML-editori.

Netscape 3.0 on kaupallinen tuote, joten sitä ei voi enää levittää vapaasti. Internetistä voi hakea FTP:llä Navigatorin beta-version, mutta sillä on tietty käyttöaika. Ajan umpeutuessa on haettava uudempi versio. Kirjan kirjoittamishetkellä Navigatorin 3.0 oli juuri julkaistu. Tämän kirjan esimerkit on testattu sillä.

Seuraavaksi tutustutaan selaimen ominaisuuksiin vain päällisin puolin, koska selaimen ominaisuuksista on jo olemassa Atk-kustannuksen kustantama kirja. Netscape 2 on käsitelty perusteellisesti kirjassa Kimmo Rousku: Netscape 2 & World Wide Web. Selaimen 3.0 versio on käyttöliittymältään samanlainen kuin 2.0 versio.



Kannattaa tutustua Netscapen Options-valikkoon, koska se sisältää tärkeitä oletusasetuksia. General Preferences-dialogissa on yleiset asetukset, kuten fontin, värien ja ulkoisten apuohjelmien asetukset. Ulkoisilla apuohjelmilla Netscapen käyttöä voi tehostaa. Apuohjelmiksi kannattaa laittaa RealAudio, VRML-laajennus ja Video for Windows. Java-ohjelmia varten ei tarvitsee asettaa erityistä apuohjelmaa, vaan Java-ohjelmat toimivat sellaisenaan 32-bittisessä Netscapessa. Netscape 3.0 -selaimesta on olemassa 16-bittinen versio, mutta se ei tue Javaa, tosin se tukee JavaScriptiä. Vain Windows 95:n tai Windows NT:n omistajat pääsevät nauttimaan Netscapen uusista ominaisuuksista täysivaltaisesti.

Network Preferences-valikosta asetetaan välimuistin (*Cache*) koko. Välimuistiin tallennetaan väliaikaisesti haetut WWW-sivut. Samassa valikossa on proxyn eli verkkovälimuistin asetukset.

Security Preferences-valikosta määrätään onko Java ja JavaScript toiminnassa. Tällä toiminnolla voi estää Java-ohjelmien ajamisen, jos niin

haluaa. Valikon komennolla Show Java Console voi puolestaan tutkia ladattujen Java-ohjelmien (virhe)ilmoituksia.

1.8 JavaScriptin mahdollisuudet

HTML-kielen avulla on tähän mennessä sivuille saatu tekstiä erilaisilla kirjasinlajeilla, interaktiivisia elementtejä painonapeilla ja listoilla. Niin hyvä kuin HTML-kieli onkin, se ei tarjoa ominaisuuksia välittömään kaksisuuntaiseen interaktiivisuuteen. JavasScriptissä on interaktiivisuuteen erinomaiset mahdollisuudet.

JavaScript-kielen avulla WWW-sivuille voidaan tehdä mitä erilaisempia sovelluksia. On mahdollista ohjelmoida esimerkiksi pelejä käyttäjien viihdyttämiseksi.

1.9 JavaScript-kieli

JavaScript on Java-kielestä kehitetty olio-pohjainen ohjelmointikieli, jonka tarkoituksena on helpottaa interaktiivisten multimedia sovellusten kehittelyä. Varsinaisen helpotuksen tuo mukanaan HTML-kielen ja JavaScript-kielen yhteiskäyttö. Javassa ei ole mahdollisuutta hyödyntää mitenkään HTML-kielen helppoutta luoda graafisia elementtejä (painonappeja, valintalistoja jne.) JavaScriptissä elementtien luonti onnistuu helposti HTML-kielen tageilla ja elementit linkitetään osaksi ohjelmaa. JavaScript-ohjelmat pystyvät vastaamaan käyttäjän tekemiin tapahtumiin, kuten hiiren napsautuksiin ja sivun valintaan. HTML-kielisiin ohjelmiin saadaan näin "älykkyyttä", koska esimerkiksi elementtien sisältämä informaatio voidaan lukea, ja palauttaa heti informaatiota käyttäjälle.

JavaScript-kielinen ohjelma on varsinaisesti osa HTML-dokumenttia, ohjelman lähdekoodi on siten nähtävissä selaimien menukomennolla view->source. JavaScript muistuttaa Java-kieltä, mutta siitä puuttuu vahva tyypitys, joka Java-kielessä on olennaista. Kieli tukee myös funktioita, kuten Java mutta ilman funktioiden määrittelyä. Tyypityksen ja funktioiden määrittelyn pois jättäminen alentaa samalla ohjelmointi kynnystä - lähes kuka tahansa voi kirjoittaa JavaScript-kielisiä ohjelmia.

JavaScript-kieltä tukevat Netscape Navigator -selaimen 2-versio ja sitä uudemmat.

1.10 Internet-osoitteita

Internetissä on tuhansia Javaa, JavaScriptiä, HTML:ää ja CGI-ohjelmia koskevia artikkeita. Seuraavassa on listattu muutamia sellaisia paikkoja, joissa jokaisen itseään kunnioittavan WWW-ohjelmoijan kannattaa käydä.

- Sunin, Javan kehittäjän sivu (**http://java.sun.com/**). Täältä löydät uusinta tietoa Javasta, voit imuroida uusimman JDK:n
- Netscapen kotisivu (**http://www.netscape.com**). Täältä löydät ajankohtaista tietoa uusista Netscape-selaimista sekä kattavat dokumentit JavaScript-kielestä ja sen kehittymisestä.
- WWW usein kysyttyjä kysymyksiä (*FAQ=Frequently Asked Questions*) (**http://sunsite.unc.edu/boutel/fag/www_fag.html**).
- CERN-palvelimen dokumentit (**http://www.w3.org/hypertext/WWW/Daemon/User/Guide.html**).
- NCSA-palvelimen dokumentit (**http://hoohoo.ncsa.uiuc.edu/**).
- NCSA-Palvelimen imurointi (**ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current**) tai suora ftp-yhteys koneeseen **ftp.ncsa.uiuc.edu** ja siellä hakemisto **/Web/httpd/Unix/nsca_httpd/current**.
- HTML-kieli (**http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html**).
- URL-osoitemekanismi (**http://www.w3.org/hypertext/WWW/Addressing/URL/Overview.html**).
- Altavista hakuropotti (**http://altavista.digital.com/**).
- Suomalainen linkkisivu *Lisko* (**http://stekt.oulu.fi/~suopanki/lisko.html**).

- Tietoa JavaScriptistä ja oppimateriaalia. (**http://www.netscape.com/comprod/products/navigator/version_3.0/javascript/index.html**), (**http://www.sapphire.co.uk/javascript/homepg.html**), (**http://water.fce.vutbr.cz/java/lstoc1.html**), (**http://www.webmagnet.com/scripting.html**).
- JavaScript OnLine Help (**http://cusk-eel.ups.edu/help/ nojava.help.html**).
- JavaScript Matikka-algoritmeja (**http://www.resortguide.com/javascript/JavaScriptMath.html**).
- JavaScript muutokset (**http://depinfo.u-bourgogne.fr/bruno/ java/ jschng.html**).
- Tietoa CGI-ohjelmista ja oppimateriaalia. (**http://hamp.hampshire.edu/~agCCS/cs/cgi-stuff.html**) , (**http://www.freenet.hamilton.on.ca/~aa437/htmlguide.html**) , (**http://www.hds.univ-compiegne.fr/WEB/rapport/cgi.html**) , (**http://www.ast.cam.ac.uk/~drtr/cgi.html**) , (**http://www.alinc.com/~jorgill/cgi-bin/cgi-ex.html**).
- CGI-kirjasto (**http://www.iserver.com/cgi/library.html**).
- CGI-skriptejä (**http://www.cdn.com/cgi.html**).
- CGI-ohjelmointi **Perl**-kielellä (**http://www.mispress.com/introcgi/online_app.html**).
- CGI-ohjelmat Windows NT:ssä (**http://www.process.com/prod/cgidesc.htp**).
- CGI-ohjelmien ympäristömuuttujat (**http://hum.amu.edu.pl/Guides/hoohoo/env.html**).
- CGI-shop (**http://lpage.com/**).
- CGI-database (**http://www.clickstream.com/dbdemo/bdbadd1.html**).
- CGI-ohjelmien parametrit (**http://brain.biap.com/datapig/mrwheat/cgi_params.html**).
- CGI-ohjelmat NCSA-palvelimessa (**http://pines6.ljcrf.edu:6530/tutorials/cgi.html**).
- GIF-animaatioita (**http://www.nbn.com/people/espresso/gif.html**).

- GIF-animaatiota + animaatioiden luonti (**http://www.hooked.net/~jpadilla/homepage.htm**).
- GIF-animaatioiden luonti (**http://www.cfw.com/~cooks/index.html**).
- Cookie-mekanismin määrittely (**http://www.netscape.com/newsref/std/cookie_spec.html**).

2. HTML-kieli

2.1 HTML-kielen perusteet

HTML-kieli (*Hypertext Markup Language*) on yksinkertainen ja selkeä kieli hyperdokumenttien kirjoittamiseen. Helppokäyttöisyyden lisäksi se on myös laiteriippumaton. Sama HTML-koodi on luettavissa kaikissa tietokonejärjestelmissä niiden laitemerkistä riippumatta.

HTML-kielen komennot sijoitetaan < ja > merkkien sisään. Nämä erottavat komennot varsinaisesta tekstistä. HTML-komennot voidaan kirjoittaa sekä isoilla että pienillä kirjaimilla. Esim. <B> ja <b> lihavointikomento tulkitaan samoin riippumatta käytetystä kirjainkoosta. Komennon vaikutuksen lopetusmerkki on /. Esim. <B>Tämä on lihavoitu</B> ja tämä ei. Edellinen näyttää lukuohjelmassa tältä.

Tämä on lihavoitu ja tämä ei

Jos lukuohjelma huomaa HTML-koodissa tuntemattoman tai virheellisen komennon, niin se ohittaa komennon välittämättä siitä. Eli jos teet virheen kirjoittaessasi dokumentin koodia, niin se ei kuitenkaan estä lukuohjelmaa lukemasta dokumenttia. Joissain tapauksissa virhe saattaa kyllä vaikuttaa dokumentin muotoiluun. Jos esimerkiksi unohdat lopettaa lihavoinnin vaikutuksen, niin kaikki teksti näkyy lihavoinnin aloituskäskystä lähtien lihavoituna.

On syytä pitää mielessä, että ***lukuohjelma hoitelee aina sivun muotoilun*** . Tämä tarkoittaa sitä, että dokumentin kirjoittaja ei voi tietää millaisella laitteistokokoonpanolla käyttäjät katselevat dokumenttia.

HTML-kielellä ei kuvata sivun ulkonäköä vaan tekstin loogista rakennetta. Tekstiin merkitään sen loogiset osat ; Otsikot, leipätekstit, sanojen korostukset ja yms. Näiden tietojen perusteella lukuohjelma muotoilee lukijan näytölle dokumentin sisällön hänen laitteistonsa mukaisesti. Tämän vuoksi dokumentissa olevat ylimääräiset välilyönnit ja kaikki rivinvaihdot tiputetaan pois lukuohjelman muotoillessa dokumenttia tulostettavaksi näytölle. Esimerkiksi seuraava teksti dokumentissa

```
Tässä      on
jotain
     tekstiä.
```

Tulostuu näytölle seuraavasti.

```
Tässä on jotain tekstiä.
```

HTML tarjoaa kuitenkin sopivat komennot rivivaihtojen ja välilyöntien tulostamiseen jos se osoittautuu tarpeelliseksi.

HTML-dokumenttien kirjoittamiseen käy mikä tahansa editori esimerkiksi Windowsissa **Notepad.** Editorin vaatimuksena on, että se tallentaa kirjoittamasi tekstin levylle puhtaana tekstinä, eli siis ilman muotoilukoodeja.

HTML-dokumentin ei tarvitse välttämättä sijaita verkossa, jotta sitä voisi tarkastella lukuohjelmalla. Useimmat lukuohjelmat osaavat lukea ja näyttää dokumentit myös levyltä. Tämä ominaisuus on eduksi dokumentteja laadittaessa. Esimerkiksi Netscapella levyllä sijaitseva dokumentti luetaan valitsemalla **File**-valikosta **Open File**. Päivitettäessä dokumenttia kannattaa valita **Reload**-painonappi, jotta muutokset tulisivat näkyviin.

HTML-dokumenttitiedostojen tarkentimeksi kannattaa antaa joko .htm tai .html, vaikka tarkennin ei ole välttämätön. Ohjelma näyttää kyllä dokumentin ilman tarkentimiakin.

2.2 HTML-komennot

Komennoissa esiintyvät valinnaiset osat on sijoitettu hakasulkeiden ([]) sisään. Komentojen parametrit on merkitty *kursiivilla*. Jos komento sisältää erilaisia vaihtoehtoja, ne on erotettu toisistaan pystyviivalla (|). HTML-komentoja sanotaan yleisesti **tag**:eiksi.

2.2.1 Ensimmäinen ja viimeinen komento

<HTML> … </HTML>

Koko dokumentin koodi sijaitsee näiden tagien välissä.

2.2.2 Otsikkotiedot

<HEAD> …. </HEAD>

Näiden tagien välissä oleva koodi latautuu muistiin ensimmäisenä. HTML-dokumentti jaetaan otsikkotietoihin ja varsinaiseen runkoon. Otsikkoalueella sijaitsevat tiedot eivät yleensä tule näkyviin lukuohjelmassa, poikkeuksena dokumentin nimi. **<head>** ja **</head>** tagien välissä voi käyttää seuraavia komentoja. **<title>**, **<base>**, **<isindex>**, **<link>**, **<meta>**, **<!-- kommentti - ->** ja **<script>** sekä niihin mahdollisesti liittyviä lopetuskomentoja.

2.2.3 Dokumentin nimi

<TITLE> *Nimi* </TITLE>

Nimi tarkoittaa lukuohjelman ikkunan yläreunaan ilmestyvää dokumentin nimeä, eikä siis tiedostonimeä.

2.2.4 Dokumentin osoite

<BASE HREF="*URL*">

Base-komento kertoo lukuohjelmalle sillä hetkellä ladattuna olevan dokumentin (www-sivun) URL-osoitteen. Yleensä lukuohjelmat tietävät tämän jo valmiiksi. **Base**-komentoa voi käyttää hyödyksi silloin jos dokumentin linkit ladataan jostain muualta kuin itse dokumentti. Seuraava **Esimerkki** lataa demo-nimisen dokumentin **base**-komennolla mainitusta osoitteesta, vaikka sen dokumentin, jossa linkki sijaitsee, osoite olisikin joku muu.

```
<HTML>
<HEAD>
<BASE HREF="http://joku.muu.kone.fi/www/">
</HEAD>
<BODY>
<A HREF="demo.html">laskin</A>
</BODY>
</HTML>
```

2.2.5 Dokumentin runko

<BODY [background="*taustakuvan URL*"] [bgcolor="#*väri*"] [text="#*väri*"] [link="#*väri*"] [vlink="#*väri*"] [alink="#*väri*"]>

Dokumentin runko </BODY>

link = linkin väri
vlink = seuratun linkin väri.
alink = aktiivisen linkin väri.
text = tekstin väri.
bgcolor = taustan väri.

Väri = esim. ”#00FF00” heksana tai ”#0,255,0”
Desimaalilukuna. Värien määrittelyt annetaan RGB-arvoina (*Red, Green, Blue*). Värit voi myös kirjoittaa suoraan (*englanniksi*), esimerkiksi bgcolor=black.

<Body> ja </body> tagien väliin sijotetaan varsinainen HTML-dokumentin koodi leipäteksteineen.

2.2.6 Omat Otsikkotiedot

<META [http-eguiv=”Otsikkokenttä”]
[name=”Otsikkokentän nimi”] content=”Tieto”>

Otsikkokenttä = Oma otsikkokenttä lähettäväksi palvelimen antamien tietojen mukana.

Otsikkokentän nimi = Itsemääritellyn otsikkokentän nimi.

Tieto = Itsemääritellyn otsikkokentän sisältämä tieto.

HTML-dokumentin otsikkotietoihin voidaan lisätä itsemääriteltyjä otsikkokenttiä. Niissä voi antaa tietoja, joiden antamiseen ei ole olemassa HTML-komentoja. Näin voi lisätä otsikkoon vaikka tekijän nimen.

<META name=”Author” content=”Jaska Jokunen”>

HTML-dokumentin mukana palvelin lähettää ennen varsinaista dokumenttia joukon otsikkokenttiä, joissa kerrotaan lukuohjelmalle tulevan tiedon tyyppi, päivitysaika jne. Näihin kenttiin voi lisätä oman otsikkokentän määrittelyn **http-equiv** avulla, esimerkiksi.

<META http-equiv=”Location” content=”Finland”>

Palvelin lisää dokumentin otsikkotietoihin seuraavan rivin.

```
Location: Finland
```

2.2.7 Ääkkoset (skandit) ja erikoismerkit

ä = ä	Ä = Ä	ö = ö
Ö = Ö	å = å	Å = Å
& = &	< = <	> = >
® = ®	© = ©	” = "
Kiinteä välilyönti =	Tavuviiva =	

Joitakin erikoismerkkejä ei saa näkymään lukuohjelmassa ilman erityistä koodausta (esimerkiksi enemmän kuin yksi välilyönti peräkkäin). Useimmat lukuohjelmat näyttävät kuitenkin nykyisin ääkköset oikein ilman erityistä koodausta.

2.2.8 Fontit

<FONT [SIZE=”*koko väliltä ±(1-7)*”]
[face=”*kirjasintyyppi*”[,”*kirjasintyyppi*”]]
[color=#”*väri*”]> *Teksti* </FONT>

face = esim. ”Arial”,”Courier New”

Väri = esim. ”#00FF00” heksana tai ”#0,255,0” desimaalilukuna. Värien määrittelyt annetaan RGB-arvoina (*Red, Green, Blue*). Värit voi myös kirjoittaa suoraan (*englanniksi*), esimerkiksi color=red.

Kirjasinkokoa ja -väriä voi vaihtaa kesken dokumentin. Kaikki kirjasinkoon muutokset tehdään suhteellisina. Oletuskoko on 3. Kokoa voi vaihtaa kirjoittamalla uuden numeron vanhan tilalle tai vähentämällä ja lisäämällä vanhaa kokoa suhteellisesti sijoittamalla sen eteen plus-merkin (+) tai miinus-merkin (-).

Jotkut lukuohjelmat pystyvät vaihtamaan myös tulostusfonttia. Esimerkiksi seuraava komento tulostuu lukuohjelmaan (Netscape 3.0) Arial- tai vaihtoehtoisesti Courier New -fontilla.

Tämä on koe

Tuloste:

Tämä on koe

2.2.9 Rivinvaihto

2.2.10 Kappale

<P [align=center]> *Teksti* </P>

2.2.11 Otsikko

<Hx [align=center]> *Otsikko* </Hx> // x = luku väliltä 1-6

Luku välitä 1-6 määrää otsikon fontin koon.

2.2.12 Väliviiva

<HR [size=*paksuus*] [size=*leveys*|*leveys prosentteina*%]
[align=center|left|right] [noshade] >

noshade = pelkkä viiva, ei varjostusta.

2.2.13 Lihavointi

<B> *Teksti* </B>

2.2.14 Kursiivi

<i> *Teksti* <i>

2.2.15 Alleviivaus

<U> *Teksti* </U>

2.2.16 Kommentti

<!-- *kommentti* - ->

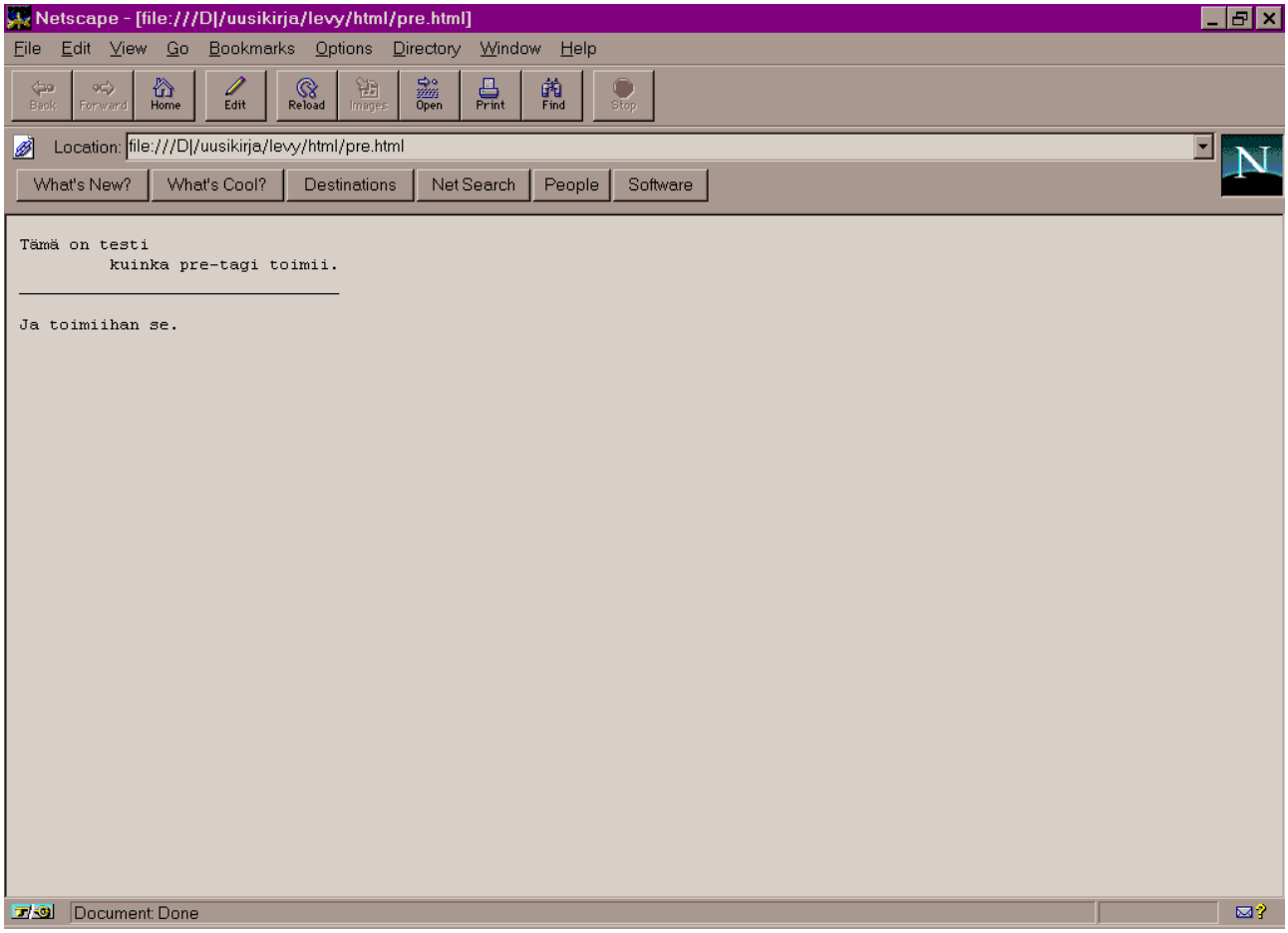
2.2.17 Esimuotoiltu Teksti

<PRE> *Teksti* </PRE>

<pre> ja </pre> tagien sisään sijoitettu teksti tulostuu lukuohjelmassa täsmälleen siten kuin se on HTML-koodiin kirjoitettu.

Esimerkki.

```
<HTML>
<BODY>
<PRE>
Tämä on testi
         kuinka pre-tagi toimii.
_____________________________________

Ja toimiihan se.
</PRE>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.2.18 Numeroitu lista

<OL [type=A|a|I|i|1] [start=numero]>

<li [type=A|a|I|i|1] [value=numero]> *Teksti*
 Teksti
 Teksti

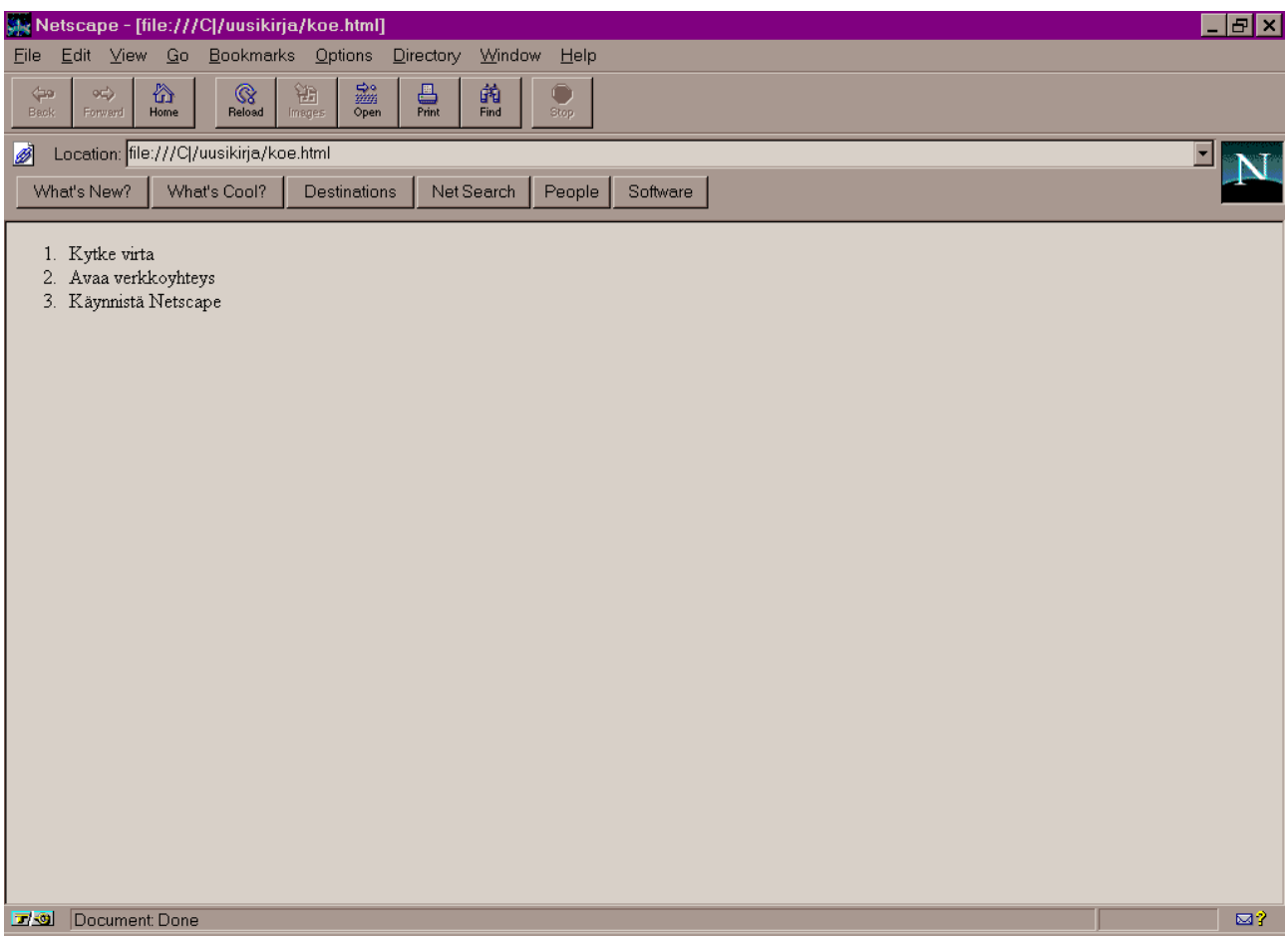
</OL>

A ja a = aakkoset
I ja i = roomalaiset numerot
1 = numerot .
start = lista aloitetaan annetusta numerosta.
value = muuttaa numerointia kesken listan. Uusi numerointi

vaikuttaa kaikkiin seuraaviin kohtiin.

Esimerkki:

```
<HTML>
<BODY>
<OL>
<LI> Kytke virta</LI>
<LI> Avaa verkkoyhteys</LI>
<LI> Käynnistä Netscape</LI>
</OL>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.2.19 Ranskalaiset viivat

<UL [type=disc|circle|square]>

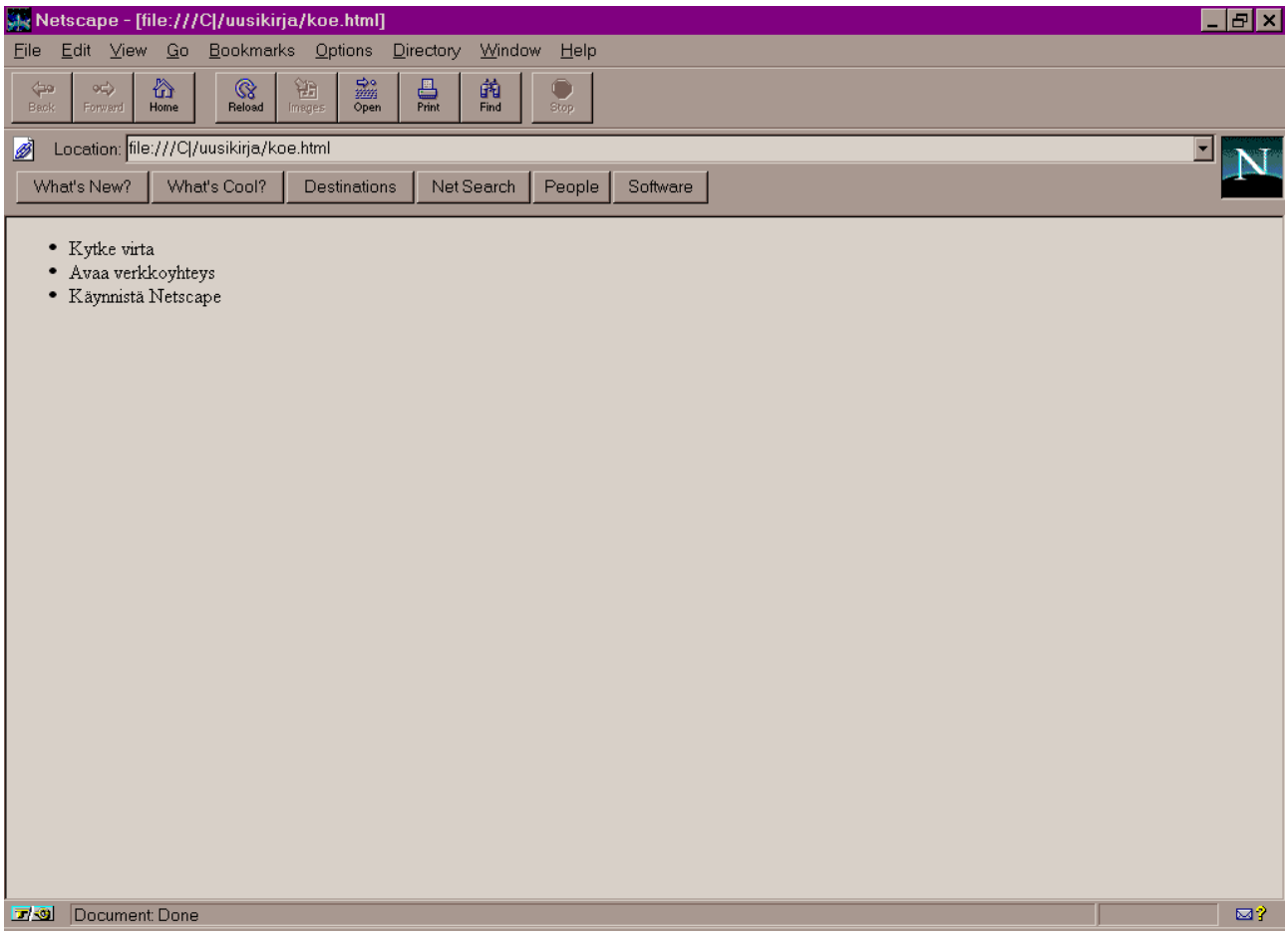
<li [type=disc|circle|square]> *Teksti*
 Teksti
 Teksti

</UL>

disc = pallo
circle = ympyrä
square = neliö .

Esimerkki:

```
<HTML>
<BODY>
<UL type=disc>
<LI> Kytke virta</LI>
<LI> Avaa verkkoyhteys</LI>
<LI> Käynnistä Netscape</LI>
</UL>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.2.20 Määritelmälistat

<DL [compact]>

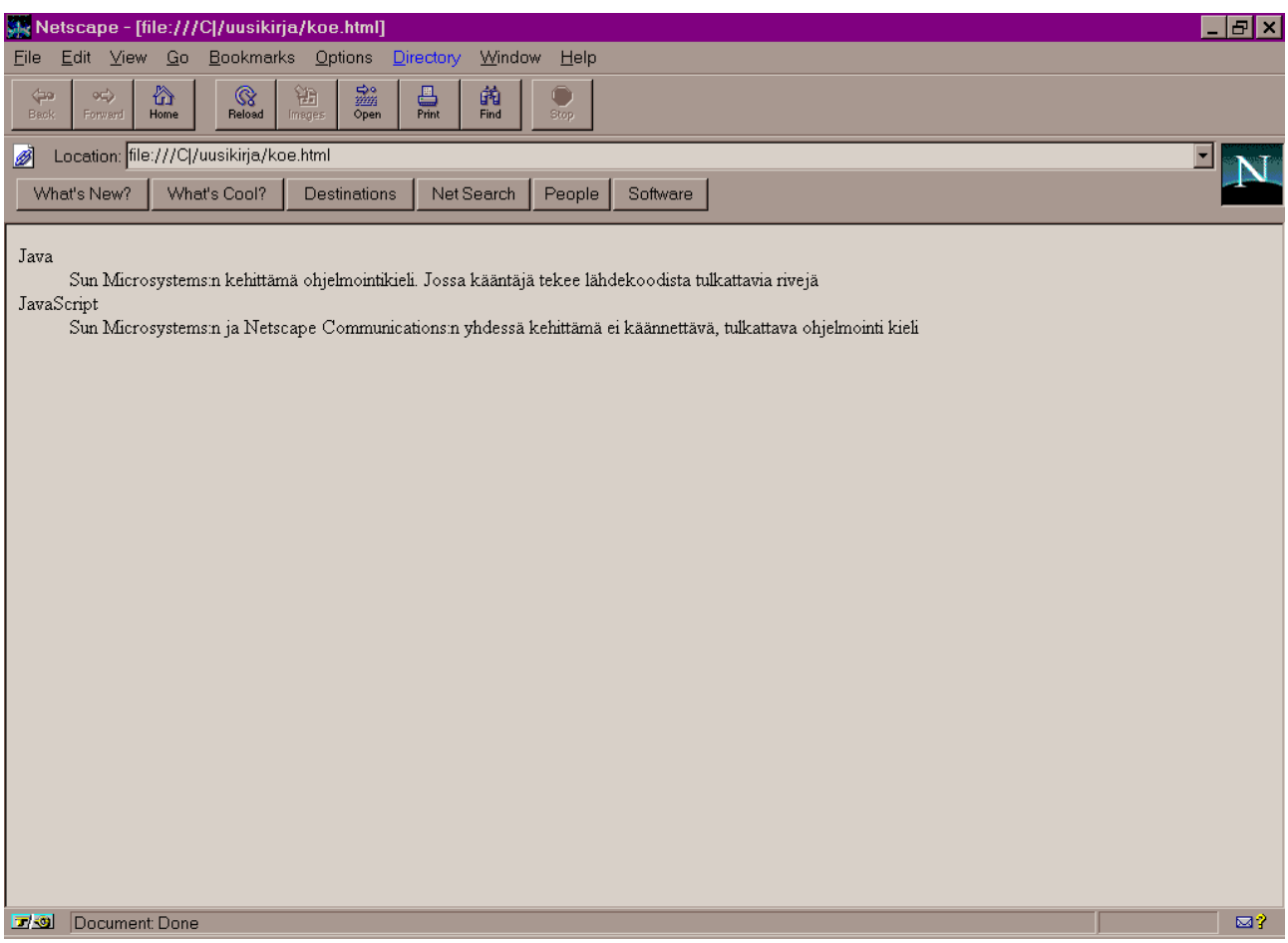
<DT> *Määrittely* </DT>
<DD> *Määritelmäteksti* </DD>
<DT> *Määrittely* </DT>
<DD> *Määritelmäteksti* </DD>

</DL>

compact = pakattu, selainohjelma pyrkii tekemään listasta pienempään tilaan sopivan.

Esimerkki:

```
<HTML>
<BODY>
<DL>
<DT>Java</DT>
<DD>Sun Microsystems:n kehittämä
ohjelmointikieli. Jossa
 kääntäjä tekee lähdekoodista tulkattavia
rivejä</DD>
<DT>JavaScript</DT>
<DD>Sun Microsystems:n ja Netscape
Communications:n yhdessä kehittämä ei
käännettävä, tulkattava ohjelmointi
kieli</DD>
</DL>
</BODY>
</HTML>
```



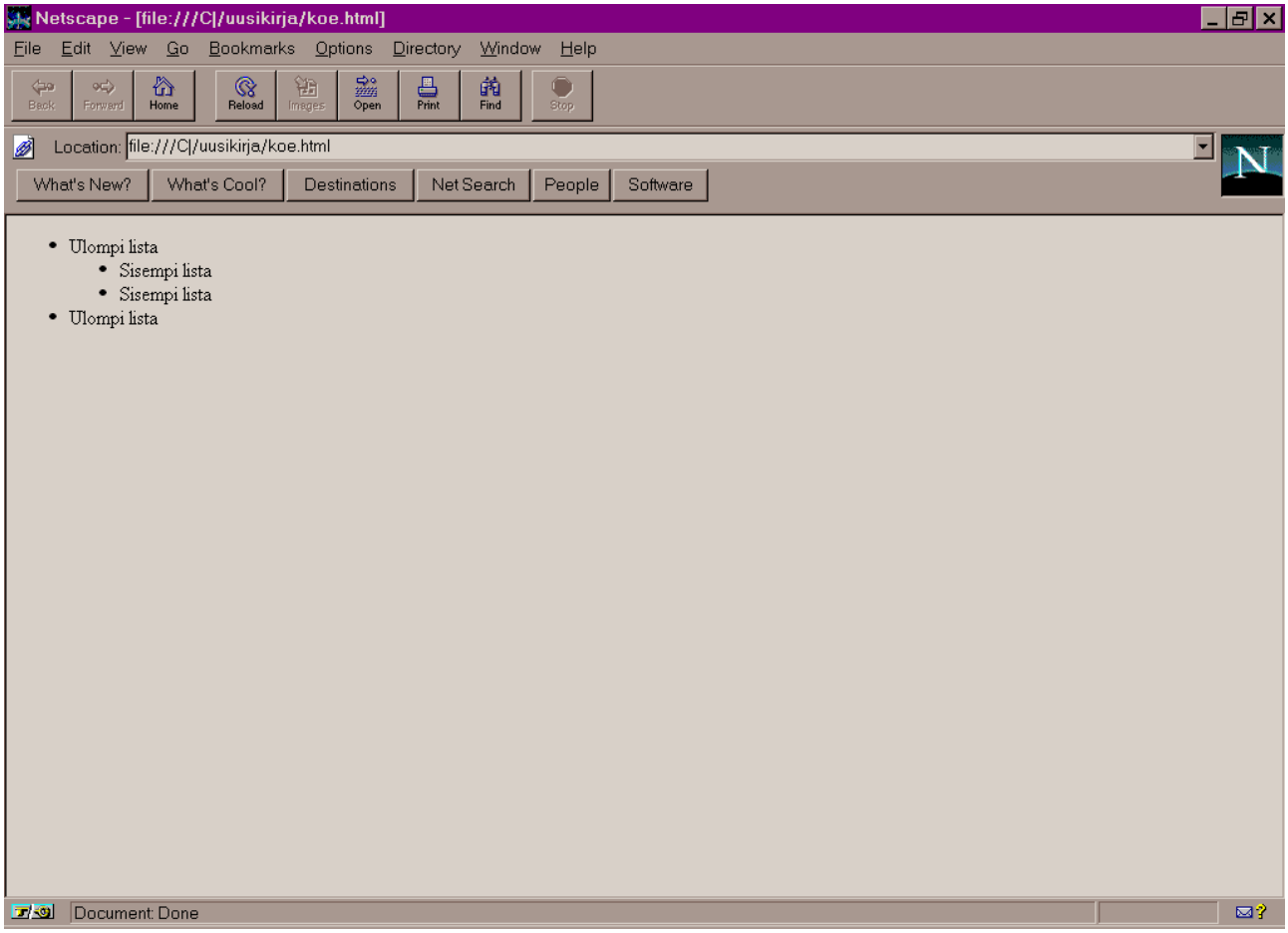
Edellisen esimerkin tuloste.

2.2.21 Sisennetyt listat

```
<UL>

<li> Teksti </li>
<UL>                    // Sisempi lista alkaa
<li> Teksti </li>
<li> Teksti </li>
</UL>                   // Sisempi lista loppuu
<li> Teksti </li>

</UL>
```

Esimerkki:

```
<HTML>
<BODY>
<UL type=disc>
<LI>Ulompi lista</LI>
<UL type=disc>
<LI>Sisempi lista</LI>
<LI>Sisempi lista</LI>
</UL>
<LI>Ulompi lista</LI>
</UL>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.2.22 Keskitys

<CENTER> *Teksti* </CENTER>

2.2.23 Rivinvaihdon estäminen

<NOBR> *Teksti* </NOBR>

<WBR>

Lukuohjelma ei sijoita rivinvaihtoa tekstiin joka on sijoitettu <nobr> ja </nobr> tagien väliin. Jos tällä välillä kuitenkin on kohta johon rivinvaihto voidaan tarpeen tullen sijoittaa, se voidaan merkitä tagilla <wbr>. Esimerkiksi seuraava lause katkaistaan lukuohjelmassa <wbr> tagin kohdalta jos rivinvaihto on tarpeen. On huomattava, että <wbr> ei välttämättä aiheuta rivinvaihtoa.

```
<NOBR>Älä katkaise tätä riviä
<WBR>mielivaltaisesta kohdasta</NOBR>
```

2.2.24 Taulukko

<TABLE[border=*f*] [width=a[%]]
[cellspacing=b][cellpadding=c]>

[<caption [align=top|bottom]> *Teksti* </caption>]
// Otsikko

[<TH [rowspan=x] [colspan=y] [valign=top|bottom|middle]
[align=left|right|center] [nowrap] [width=a[%]] > *Teksti* </TH>]
// Otsikkosolu
<TR [valign=top|bottom|middle] [align=left|right|center] >
// Rivit

<TD [valign=top|bottom|middle] [align=left|right|center]
[nowrap] [rowspan=x] [colspan=y] [width=a[%]]> *Teksti*
// Solut

</TD></TR></TABLE>

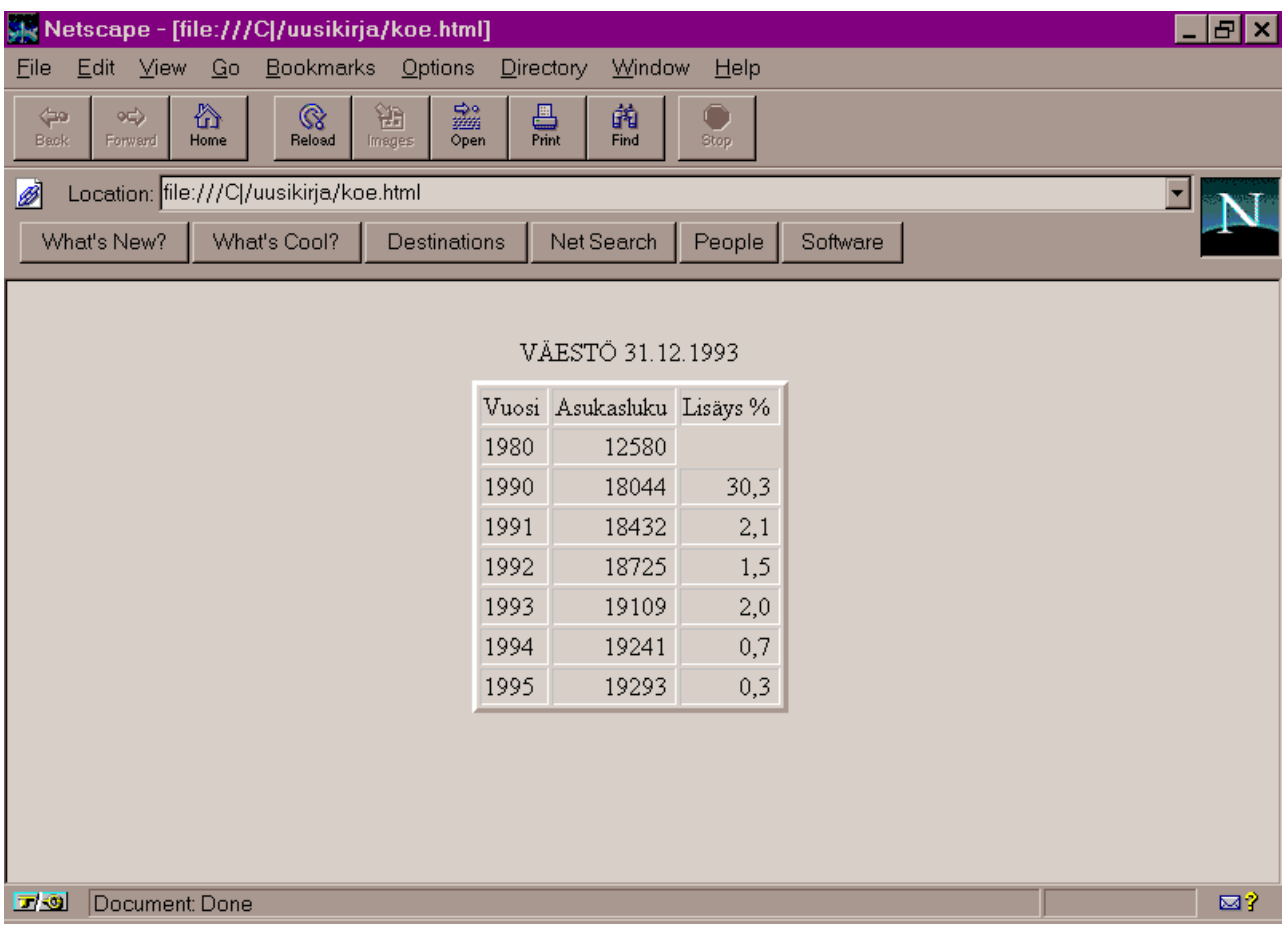
nowrap = solun tekstiä ei katkaista välilyönnin kohdalta, vaikka taulukko ei mahtuisi selainohjelman ruudulle.

a = leveys pikseleinä tai prosentteina.
b = solujen välinen tila pikseleinä.
c = solun seinämän ja tekstin välinen tila pikseleinä.
f = reunus pikseleinä.
x ja y = solun viemä tila riveinä ja sarakkeina.

Esimerkki:

```
<HTML>
<BODY>
<CENTER>
<TABLE border=”3”>
<CAPTION>VÄESTÖ 31.12.1993</CAPTION>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>Vuosi
<TD ALIGN=left VALIGN=bottom COLSPAN=2>Asukasluku
<TD ALIGN=left VALIGN=bottom COLSPAN=2>Lisäys %

<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1980
<TD ALIGN=right VALIGN=bottom COLSPAN=2>12580
<TD ALIGN=right VALIGN=bottom COLSPAN=2>

<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1990
<TD ALIGN=right VALIGN=bottom COLSPAN=2>18044
<TD ALIGN=right VALIGN=bottom COLSPAN=2>30,3
<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1991
<TD ALIGN=right VALIGN=bottom COLSPAN=2>18432
<TD ALIGN=right VALIGN=bottom COLSPAN=2>2,1

<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1992
<TD ALIGN=right VALIGN=bottom COLSPAN=2>18725
<TD ALIGN=right VALIGN=bottom COLSPAN=2>1,5

<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1993
<TD ALIGN=right VALIGN=bottom COLSPAN=2>19109
<TD ALIGN=right VALIGN=bottom COLSPAN=2>2,0

<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1994
```

```
<TD ALIGN=right VALIGN=bottom COLSPAN=2>19241
<TD ALIGN=right VALIGN=bottom COLSPAN=2>0,7

<TR>
<TD ALIGN=left VALIGN=bottom COLSPAN=2>1995
<TD ALIGN=right VALIGN=bottom COLSPAN=2>19293
<TD ALIGN=right VALIGN=bottom COLSPAN=2>0,3
</TABLE>
</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.2.25 Linkit

```
<A HREF="URL">Linkin teksti</A>
```

Tai Linkkinä kuva.

```
<A HREF="URL"><IMG SRC="URL"> </A>
```

Linkki samalle sivulle (kirjanmerkki).

```
<A HREF="#hyppäyskohta">Linkin teksti tai kuva</A>
<A NAME="hyppäyskohta">Tekstiä</A>
```

URL (*Uniform Resource Locator*) on osoitustapa verkossa sijaitsevaan tiedostoon. Se on useimmiten seuraavaa muotoa.

```
http://www.kone.maa./polku/tiedosto.html
```

Ensimmäinen osa on käytettävä protokolla (yllä **http**). Seuraavaksi kerrotaan koneen nimi internet-muodossa (myös IP-osoite numerona käy) erotettuna protokollasta kaksoispisteellä ja kahdella kauttaviivalla (yllä **www.kone.maa**). Seuraavaksi kerrotaan tiedostoon johtava polku palvelimessa erotettuna kauttaviivalla koneen nimestä (yllä **polku**). Viimeiseksi kerrotaan haettavan tiedoston nimi erotettuna kauttaviivalla polusta (yllä **tiedosto.html**).
Jos haettava tiedosto on samassa osoitteessa kuin dokumentti, jolla linkki sijaitsee, ei tarvitse käyttää protokolla eikä polku osaa. Esimerkiksi.

```
<A HREF="tiedosto.html"> Linkin teksti</A>
```

URL:ssä sallitaan ainoastaan ASCII-merkkejä. Lisäksi välilyönti ja prosentti (%) on varattu erityistarkoituksiin. Näitä ja muita ASCII-merkkejä jotka eivät ole normaalisti tulostettavissa voidaan esittää merkkikoodina. Tämä tapahtuu siten, että selvitetään halutun merkin ASCII-koodi heksalukuna ja sijoitetaan tämän luvun eteen prosenttimerkki (%). Esimerkiksi ”matomerkin” (~) ASCII-koodi heksana on 7E ja näin ollen se voidaan koodata URL:ään koodilla %7E.
Seuraavia merkkejä kutsutaan epävarmoiksi merkeiksi ja niitä on syytä välttää URL:ssä.

”;”, ”/”, ”?”, ”:”, ”@”, ”=”, ”#”, ”<”, ”>”, ”&” ja lainausmerkki

---

2.2.26 Indeksitiedosto

Indeksitiedosto on dokumentti, jonka palvelinohjelma lähettää lukuohjelmalle, jos lukuohjelman pyytämässä URL:ssä ei ole mainittu tiedoston nimeä, vaan pelkkä polku. Indeksitiedoston nimeksi tulee antaa **index.html**, jollei palvelimen määrittelytiedostoissa ole määrätty jotakin toista nimeä. Jos hakemisto, johon URL osoittaa, ei sisällä index.html-nimistä dokumenttitiedostoa, niin palvelin lähettää lukuohjelmalle osoitetun hakemiston tiedostolistauksen. Tämä toiminto voidaan myös estää palvelimen määrittelytiedostoissa.
Esimerkiksi kotisivun nimeksi voidaan antaa index.html ja osoitteeksi riittää näin ollen pelkkä protokolla ja polkuosa.

http://www.kotikone.fi/

2.2.27 Automaattisesti päivittyvä dokumentti

Normaalisti dokumentti näkyy lukuohjelmassa sellaisena kuin se taushetkellä oli ja jos käyttäjä haluaa nähdä onko dokumentti muuttunut, pitää se suorittaa **reload**-toiminnolla. Dokumenttiin voidaan lisätä tieto, joka kertoo lukuohjelmalle, että se pitää ladata uudelleen määräajoin. Tämä toiminto hoidetaan **meta**-komennon avulla seuraavasti.

```
<HTML>
<HEAD>
<META http-equiv=”Refresh” content=”60”>
</HEAD>
```

Määrittelyyn **http-equiv** annetaan otsikkokentäksi ”*Refresh*” ja määrittelyyn **content** sen arvoksi *päivitystaajuus* (tässä esimerkissä 60 sekuntia).
Tällä menetelmällä voidaan ladata päivitysajan umpeutuessa myös joku muu kuin alkuperäinen dokumentti pienellä lisäyksellä.

```
<HTML>
<HEAD>
<META http-equiv=”Refresh” content=”60;
URL=http://www.kone.fi/polku/tiedosto.html”>
</HEAD>
```

Tässä päivitysajan jälkeen lisätään puolipiste ja tyhjä välilyönti sekä määritelmä ”*URL*” ja osoite, josta uusi dokumentti ladataan kun päivitysaika on kulunut. Jos uusi dokumentti ei sisällä enää päivityskomentoa, jää lukuohjelma odottamaan käyttäjän toimenpiteitä.

2.2.28 Kuvat

```
<IMG SRC=”URL”

[align=left|right|texttop|absmiddle|baseline|absbottom]

[vspace=”pystymarginaali pikseleinä”]
[hspace=”vaakamarginaali pikseleinä”]

[alt=”tekstivaihtoehto”] [border=”reunus pikseleinä”]

[width=”leveys pikseleinä[%]”] [height=”korkeus pikseleinä[%]”]>

<BR [clear=left|right]>
```

Aiheuttaa kuvan vierellä kulkevan tekstin tulostumaan vasta kuvan jälkeen. (Kuva sijaitsee joko tekstin vasemmalla tai oikealla puolella).

align=right, kuva sijoitetaan kuvaruudun oikeaan reunaan ja sitä seuraava teksti kuvan vasemmalle puolelle.

align=left, kuva sijoitetaan kuvaruudun vasempaan reunaan ja sitä seuraava teksti kuvan oikealle puolelle.

align=texttop, sijoittaa kuvan yläreunan samalle tasolle tekstin ylimmän rivin kanssa.

align=absmiddle, sijoittaa kuvan keskikohdan samalle tasolle tekstirivin keskikohdan kanssa.

align= absbottom, sijoittaa kuvan alareunan samalle tasolle tekstin alimman rivin kanssa.

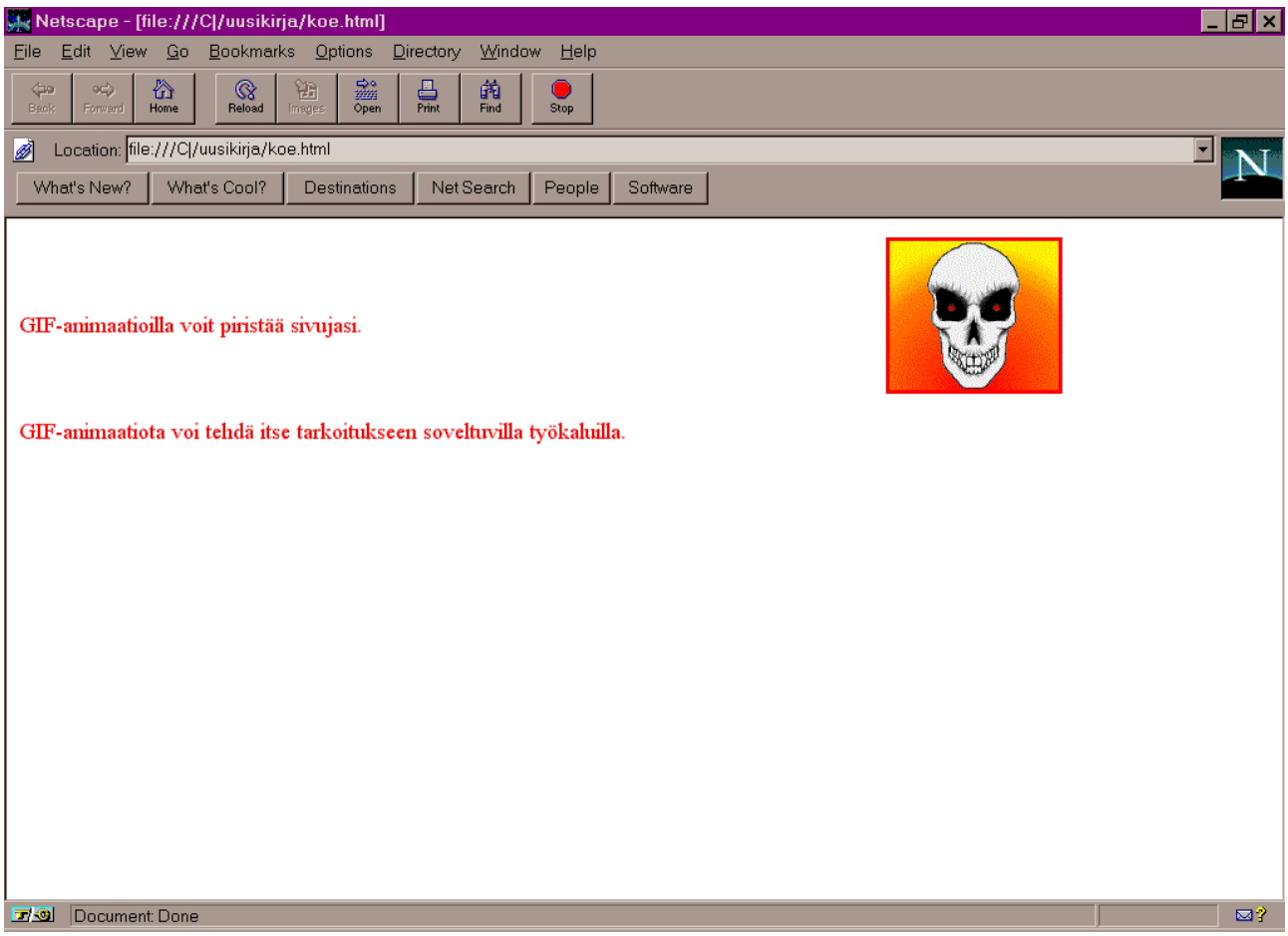
align= bottom, sijoittaa kuvan alareunan tekstirivin perusviivan tasolle.

alt = Vaihtoehtoinen teksti, jos selainohjelma ei pysty näyttämään kuvaa.

Esimerkki:

```
<HTML>
<BODY bgcolor="#ffffff">
<FONT ="#ff0000" size="4">

<IMG SRC="animskul.gif" align=right
border="3" hspace=150>

<BR><BR><BR>
GIF-animaatioilla voit piristää sivujasi.
<BR clear=right>
<BR>GIF-animaatiota voi tehdä itse tarkoitukseen
```

```
soveltuvilla työkaluilla.
      </FONT>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

Läpinäkyvällä taustalla (*Transparent color*) tarkoitetaan sitä, että yksi kuvan väreistä on määritelty taustaväriksi, joka värittyy aina samanväriseksi kuin dokumentin taustaväri. Määrittely tehdään jollakin kuvankäsittelyohjelmalla (esim. Lviewpro 32). Määrittely on olemassa vain **GIF89**-formaatissa.

Interlace muodossa talletettu kuva eroaa tavallisesta siinä, että se latautuu ikään kuin kerroksittain lukuohjelmaan, eikä riveittäin kuten tavallinen kuva. Tämän ansiosta kuva näkyy jo hyvin aikaisessa latausvaiheessa epäselvänä hahmona. Myös tämä ominaisuus on olemassa **GIF89**-formaatissa ja talletus tähän formaattiin tehdään jollakin kuvankäsittelyohjelmalla (esim. Lviewpro 32).

2.2.29 Kehykset (Frames)

Lukuohjelman tulostusikkuna on mahdollista jakaa myös useampiin itsenäisiin osiin, joista tässä käytetään nimitystä kehykset. Itsenäisyys tulee siitä, että jokaiseen kehykseen voidaan ladata eri sisältö (HTML-dokumentti). Kehyksissä voi olla myös vierityspalkit Windows-ikkunoiden tapaan. Kehyksiä käytettäessä korvataan BODY-tagi FRAMESET-tagilla. Lisäksi tarvitaan FRAME-tagi varsinaisia kehyksiä muodostamaan. Kehykset muodostetaan seuraavasti.

```
<HTML>
<FRAMESET cols="sarakeleveys pikseleinä [%] [*]" rows="rivit
pikseleinä [%] [*]">

<FRAME SRC="URL" [NAME="kehyksen nimi"]
[MARGINWIDTH="arvo pikseleinä" ]
[MARGINHEIGHT="arvo pikseleinä" ] [SCROLLING="yes|no|auto"]
[NORESIZE]

</FRAMESET>
</HTML>
```

Sarakkeiden ja rivien arvot voidaan antaa pikseleinä tai prosentteina pilkulla (,) eroteltuna. Käytännössä prosentteina annetut arvot toimivat parhaiten, koska käyttäjän laitteistokokoonpano ei yleensä ole tiedossa. Sarakkeiden, kuten myös rivien, yhteismäärä pitää olla 100%. Tähtimerkkiä (*) käytetään antamaan kehykselle kaikki jäljellä oleva tila. Tähtimerkin eteen voidaan sijoittaa myös luku, esimerkiksi.

```
<FRAMESET cols="2*,*" >
```

Tämä ilmaisu antaa ensimmäiselle sarakkeelle kaksikertaa niin paljon tilaa kuin toiselle.

```
<FRAMESET rows="20%,60%,20%">
```

Ylläolevassa esimerkissä muodostetaan kolme riviä, joista ensimmäinen ja viimeinen saavat kukin 20% tilasta ja keskimmäinen 60% tilasta.

```
<FRAMESET rows="100,*,100">
```

Ylläolevassa esimerkissä muodostetaan kolme riviä, joista ensimmäinen ja viimeinen on määrätty 100 pikselin levyisiksi ja keskimmäiselle jää lopputila käytettäväksi.

Kehykselle kannattaa antaa yleensä nimi, koska sen avulla voidaan eri tulosteita ohjata juuri tiettyyn kehykseen.

Marginaalin leveys pysty- ja vaakasuunnassa voidaan asettaa komennoilla **MARGINHEIGHT** ja **MARGINWIDTH**.

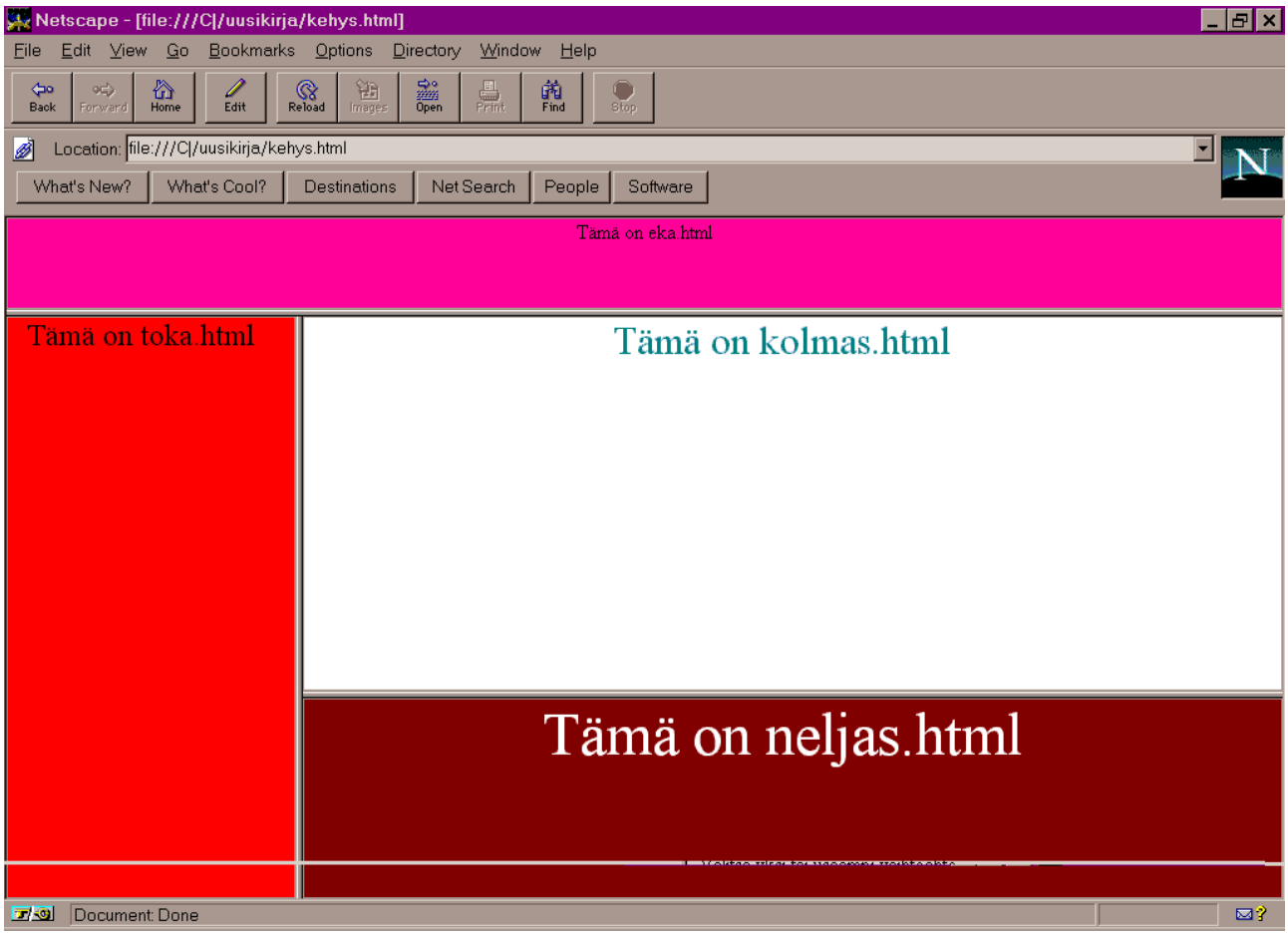
Vierityspalkit voidaan asettaa päälle/pois tai automaattisiksi (ilmestyvät jos siihen on aihetta) **SCROLLING** komennolla. Oletusarvo on auto.

Jos halutaan, että kehyksien kokoa ei voida muuttaa, otetaan käyttöön **NORESIZE** komento, muuten se jätetään pois. Oletusarvoisesti kaikkien kehysten koko on muuteltavissa.

Lisäksi voidaan käyttää **<NOFRAMES> </NOFRAMES>** -tageja, joilla näytetään tagien välinen sisältö lukuohjelmille, jotka eivät tunnista kehyksiä. Lukuohjelma, joka tunnistaa kehykset jättää huomioimatta näiden tagien väliin jäävän osan dokumentista.

Esimerkki kehyksien käytöstä. Dokumentit eka.html, toka.html, kolmas.html ja neljas.html pitää tehdä erikseen.

```
<HTML>

<FRAMESET ROWS="80, *">

<FRAME SCROLLING="no"  NAME="eka"  SRC="eka.html"
        MARGINWIDTH=0  MARGINHEIGHT=0>

<FRAMESET COLS="240, *">

<FRAME SCROLLING="auto"  NAME="toka"  SRC="toka.html"
               MARGINWIDTH=0  MARGINHEIGHT=0>

<FRAMESET ROWS="65, 35">

<FRAME SCROLLING="auto"  NAME="kolmas"  SRC="kolmas.html"
MARGINWIDTH=0  MARGINHEIGHT=0>

<FRAME SCROLLING="auto"  NAME="neljas"  SRC=”neljas.html"
MARGINWIDTH=0  MARGINHEIGHT=0>

</FRAMESET>
</FRAMESET>
</FRAMESET>

</HTML>
```



Edellisen esimerkin tuloste.

2.3 Interaktiiviset kuvat

Interaktiiviset kuvat vaativat aina toimiakseen WWW-palvelimen, johon on asennettu **imagemap**-ohjelma. Ohjelma on asennettava palvelimen cgi-ohjelmahakemistoon. Ohjelma saa hiiren näpäytyksen jälkeen lukuohjelmalta kuvan kyseisen kohdan koordinaatit (pikseleinä) ja vertaa niitä karttatiedostoon, jonka perusteella se valitsee seuraavaksi näytettävän dokumentin. Kuvien pisteiden koordinaatit saa selville jollain kuvankäsittelyohjelmalla. Karttatiedosto on tavallinen tekstitiedosto, jonka voi sijoittaa samaan hakemistoon kuin itse interaktiivisen kuvan. Tarkentimena on hyvä käyttää vaikka ***.map** liitettä. Karttatiedostossa kerrotaan kuvan kuumien (eli siis kohdat, joita näpäyttämällä tapahtuu jotakin) kohtien muoto, sijainti, koko ja linkin kohde.

Tausta voidaan ilmoittaa komennolla **default** URL. Tämän rivin määrittelemä linkki valitaan silloin kun näpäytetään kuumien alueiden ulkopuolelle. Linkiksi valitaan yleensä kuva itse, jolloin 'ohi' näpäyttäminen ei vaikuta näennäisesti mitenkään. Default-määritelmän tulee olla karttatiedoston viimeisenä rivinä. Jos käytetään magneettisia pisteitä default-komentoa ei tarvita.

Nelikulmio:

rect URL x1,y1 x2,y2
Koordinaatit vasen yläkulma (x1,y1) ja oikea alakulma (x2,y2).

Ympyrä:

circle URL xk,yk xp,yp
Koordinaatit keskipiste (xk,yk) ja joku kehällä oleva piste xp,yp).

Monikulmio:

poly URL x,y [x,y] …
Koordinaatteina ilmoitetaan alueen jokaisen kulman koordinaatit. Koordinaatteja voidaan antaa korkeintaan 100 kappaletta.

Magneettiset pisteet:

point URL x,y

Magneettiset pisteet ovat kuvaan määriteltyjä pisteitä, jotka 'kaappaavat' jokaisen kuumien kohtien ulkopuolelle osuneen näpäytyksen. Jos pisteitä on useampia niistä valitaan aina näpäytystä lähinnä oleva.

Magneettiselle pisteelle määritellään yksi koordinaatti, joka osoittaa sen sijainnin.

Valmis karttatiedosto:

```
rect http://kone/kohde_1.html 12,50 100,150
circle http://kone/kohde_2.html 80,200  60,220
poly http://kone/kohde_3.html 213,759 352,587 546,750
default http://kone/tausta.html
```

Jos kohtia on määritelty päällekkäin niin suoritusjärjestys määräytyy sen mukaan mikä määrittely on kirjoitettu aikaisemmin karttatiedostoon, siis ylin rivi ensin jne.
Karttatiedosto kannatta tehdä vaikka verkosta saatavalla **Mapedit**-ohjelmalla, jolloin syntaksit tulevat automaattisesti oikein.

2.3.1 Interaktiivinen kuva dokumentissa

```
<A HREF=”imagemap karttatiedosto”>
<IMG SRC=”http://kone/kuva.gif” ISMAP>
</A>
```

imagemap = Palvelimen imagemap-ohjelman sijainti (URL)
karttatiedosto = Kuvaan liittyvän karttatiedoston sijainti
(URL:n polku-osa ja tiedoston nimi)

ISMAP-komennolla kerrotaan, että kuvaa käytetään karttatiedoston kuvana.

Esimerkki: Karttatiedosto sijaitsee käyttäjän kotihakemistossa.

```
<A        HREF="http://impedanssi.spt.fi:8998/cgi-
bin/imagemap/~emi/ptm_lin.map">

<IMG SRC="yelB2_2.gif" border="0" ismap>
</A>
```

2.4 Protokollat

2.4.1 Http

Http (*HyperText Transfer Protocol*) protokollaa käytetään hypertext-dokumenttien siirtämiseen verkossa.

http://kone[IP numero]:[porttinumero]/[polku/[tiedostonimi]] [?parametrit]

Jos porttinumero puuttuu, oletusarvona käytetään numeroa 80. Parametrinä annettu merkkijono välittyy cgi-ohjelmalle parametrina.

2.4.2 Ftp

Ftp (*File Transfer Protocol*) protokollaa käytetään kaikenlaisten tiedostojen siirtämiseen verkossa.

ftp://[käyttäjätunnus[:salasana]@]kone [:portti]/[polku/] [tiedostonimi] [;typecode=a|i|d]

Portti = ftp-palvelimen portti
typecode = määrittelee haettavan tiedoston tyypin
a = ASCII
i = binaarinen
d = tiedostolistaus (oletusarvona)

Luku ohjelma osaa ottaa yhteytta anonyymi-palvelimiin ilman tunnuksia.

2.4.3 News

News-protokollaa käytetään uutisryhmien siirtämiseen. Lukuohjelman asetuksissa on yleensä määriteltävä uutispalvelin asianomaiseen kohtaan.

news:uutisryhmä

Esimerkki: `news:sfnet.atk.mak`

nntp://kone[:portti]/uutisryhmä/artikkeli

portti = portti, johon uutispalvelin on kytketty.
artikkeli = artikkelin järjestysnumero uutisryhmässä, numerot ovat palvelinkohtaisia.

2.4.4 Telnet

Telnet-protokollaa käytetään verkossa pääteyhteyden ottamiseen koneesta toiseen. Lukuohjelmassa täytyy asetuksiin määritellä tähän tarkoitukseen sopiva ohjelma, jonka lukuohjelma yhteyttä pyydettäessä käynnistää.

telnet://[kayttäjätunnus[:salasana]@]kone[:portti]/

portti = telnetd:n palvelinohjelman käyttämä portti.

2.4.5 Mailto

Mailto-protokollalla hoidetaan sähköpostin välitys lukuohjelman ja verkon välillä.

mailto:sähköpostiosoite

sähköpostiosoite internetmuodossa.

Esimerkki. `mailto:emi@impedanssi.spt.fi`

2.4.6 Wais

Wais-protokollaa käytetään tietokantahakujen siirtoon verkossa.

wais://kone[:portti]/tietokanta[?haku]
wais://kone[:portti]/tietokanta/tyyppi/hakupolku

portti = wais-palvelimen käyttämä portti, oletusarvo = 210.
tietokanta = tietokannan nimi.
haku = tietokannan hakuehdot.
tyyppi = WAIS-dokumentin tyyppi.
hakupolku = WAIS-dokumenttiin osoittava polku.

2.4.7 File

File-protokollalla osoitetaan yhteen tiettyyn tiedostoon.

file://kone/polku/[tiedosto]
file://localhost/levy|/polku/[tiedosto]
file:///levy|/polku/[tiedosto]
levy = kiintolevyn tunnuskirjain

Esimerkki. Luetaan c-levyn temp-hakemistosta koe.html -niminen tiedosto lukuohjelmaan.

```
file:///c|/temp/koe.html
```

2.5 Mime-tyypit

Kun palvelinohjelma lähettää lukuohjelmalle tiedoston, se liittää mukaan myös tiedon tiedoston tyypistä. Jos tiedosto on hypertekstidokumentti (html-tiedosto), lukuohjelma osaa näyttää sen sellaisenaan. Jos tiedoston tyyppi on joku muu, esimerkiksi video (*.avi) lukuohjelmaa pyrkii käynnistämään ohjelman, joka osaa näyttää videon.
Tietoa, josta lukuohjelma tunnistaa tiedoston tyypin, kutsutaan **MIME**-tyypiksi. Se sisältää kaksi kauttaviivalla erotettua osaa, joista ensimmäinen määrittelee tiedon lajin ja toinen tiedon tarkemman tyypin. Seuraavassa on lueteltu eri MIME-tyyppejä.

application/octet-stream	Binääristä dataa.
application/postscript	Postscript-tiedosto.
application/zip	Zip-muotoon pakattu tiedosto.
application/tar	Tar-paketti.
audio/basic	au- tai snd-muotoa oleva äänitiedosto.
audio/x-aiff	aiff-äänitiedosto.
audio/x-wav	wav-äänitiedosto.
image/gif	gif-kuva.

image/jpeg	jpeg-kuva.
image/tiff	tiff-kuva.
text/html	HTML-tiedosto.
text/plain	ASCII-muotoinen tekstitiedosto.
video/mpeg	mpeg-video.
video/quicktime	quicktime-video.
video/x-msvideo	avi-video.

2.6 Lomakkeet (Forms)

Dokumenttiin voidaan HTML:n avulla kirjoittaa erilaisia interaktiivisia lomakkeita, joilla voidaan toteuttaa mitä moninaisimpia toimintoja. Lomake voi sisältää tekstikenttiä, painonappeja ja valikoita. Lomakkeisiin kuuluu myös olennaisena osana itse lomakkeen lisäksi myös ohjelma joka käsittelee siltä saadut tiedot ja suorittaa käsittelyn jälkeen määrätyt toimenpiteet, esimerkiksi lähettää sähköpostia tiettyyn osoitteeseen. Tähän tarkoitukseen on saatavissa valmiita ohjelmia ja itse tehtyjä käsitellään CGI-ohjelmat -kappaleessa. Lomake muodostetaan seuraavasti.

```
<FORM method="lähetystapa" [action="käynnistettävän ohjelman URL"] [enctype="lähetysformaatti"]>

Lomakkeen sisältö

</FORM>
```

Lähetystapa on joko **GET** tai **POST**, joista GET on oletusarvona. Tavassa GET saa vastaanottava ohjelma lomakkeen sisällön parametrikseen ja tavassa POST lomakkeen sisältö siirretään yhteen merkkijonoon palvelinkoneen **stdin**-puskuriin, josta se on luettavissa ohjelmointikielestä

riippuen eri tavoin. Lähetystapoja käsitellään myöhemmin lisää CGI-ohjelmien yhteydessä.

Lähetysformaattia (MIME-tyyppi) ei yleensä tarvitse erikseen antaa.

Lomake lähettää käsittelyohjelmalle merkkijonon, joka koostuu nimi-arvopareista eroteltuna et (&) -merkillä, esimerkiksi.

```
eka=ON&toka=&kolmas=ON&neljas=ON
```

<FORM> ... </FORM> tagien väliin kootaan varsinainen lomake sisältöineen.

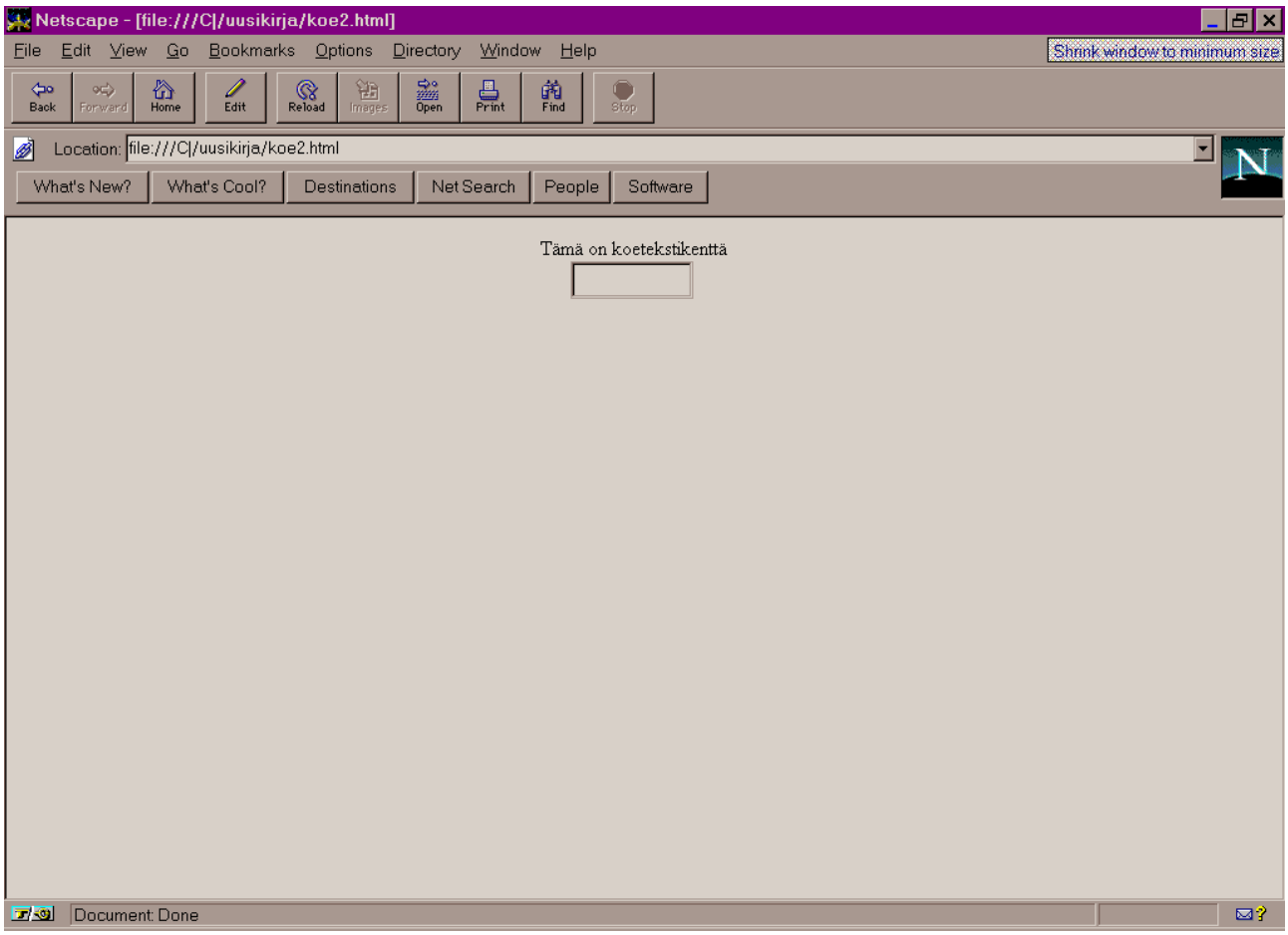
2.6.1 Tekstikentät

Tekstikenttien kautta välitetään tietoa ja ne muodostetaan seuraavasti.

<INPUT type="text" [name="*nimi*"] [maxlength="*maksimipituus*"] [size="*kentän koko*"] [value="*oletus/alkuarvo*"]>

Esimerkki.

```
<HTML>
<BODY>
<FORM method="post"
action="http://www.kone.fi/cgi-bin/ohjelma">

<CENTER>Tämä on koetekstikenttä<BR>
<INPUT type="text" name="eka" size="10">
</CENTER>
</FORM>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.6.2 Tekstialue

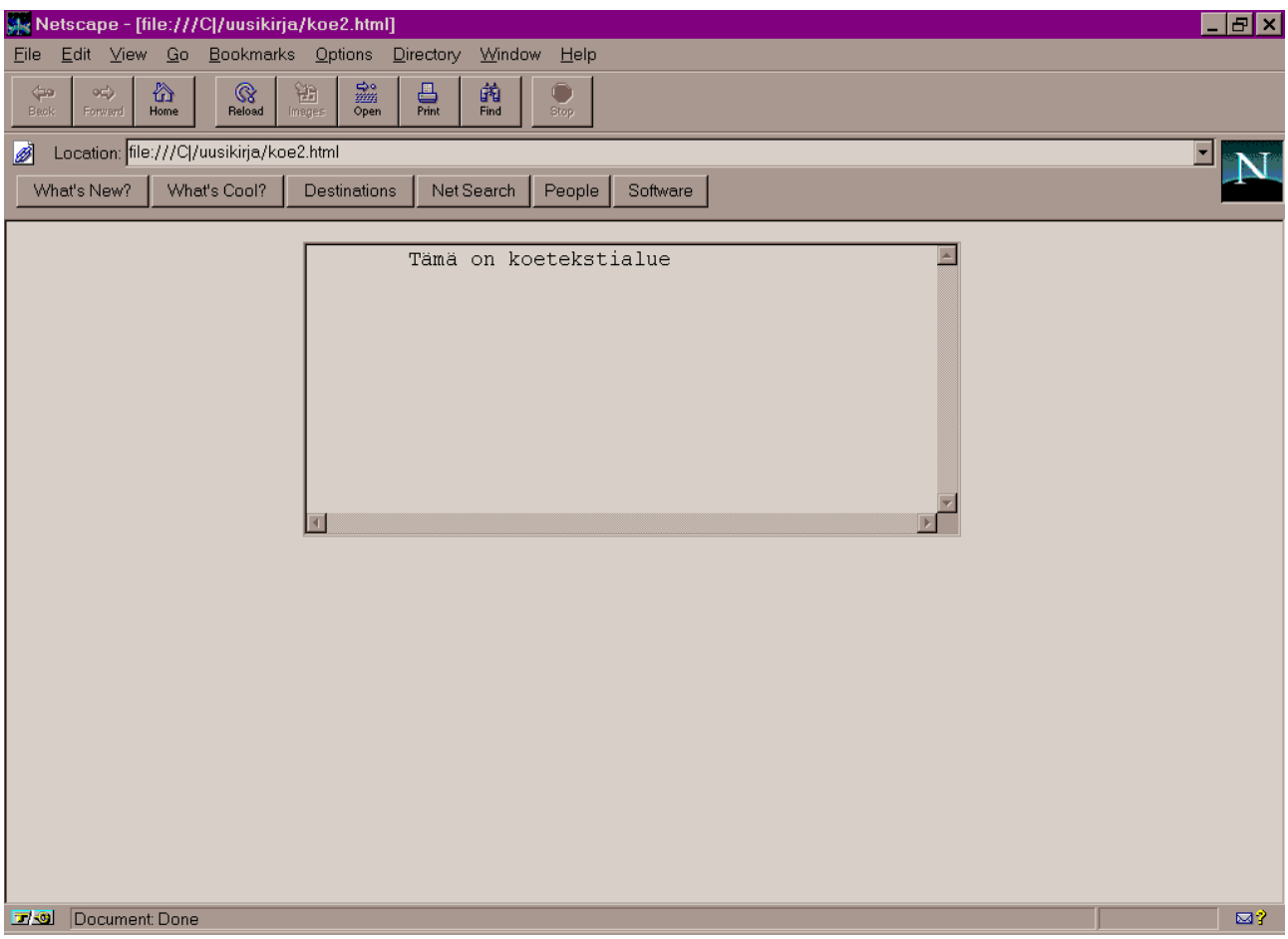
Tekstialue on periaatteessa vain suuri tekstikenttä ja sen muodostus on seuraava.

<TEXTAREA name="*nimi*" rows="*rivien lukumäärä*" cols="*sarakkeiden (pystyrivien) lukumäärä*">

Oletus/alkuarvo
</TEXTAREA>

Esimerkki.

```
<HTML>
<BODY>
<FORM method="post"
action="http://www.kone.fi/cgi-bin/ohjelma">

<CENTER>
<TEXTAREA name=”eka” rows=”10” cols=”50”>
Tämä on koetekstialue
</TEXTAREA>
</CENTER>
</FORM>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

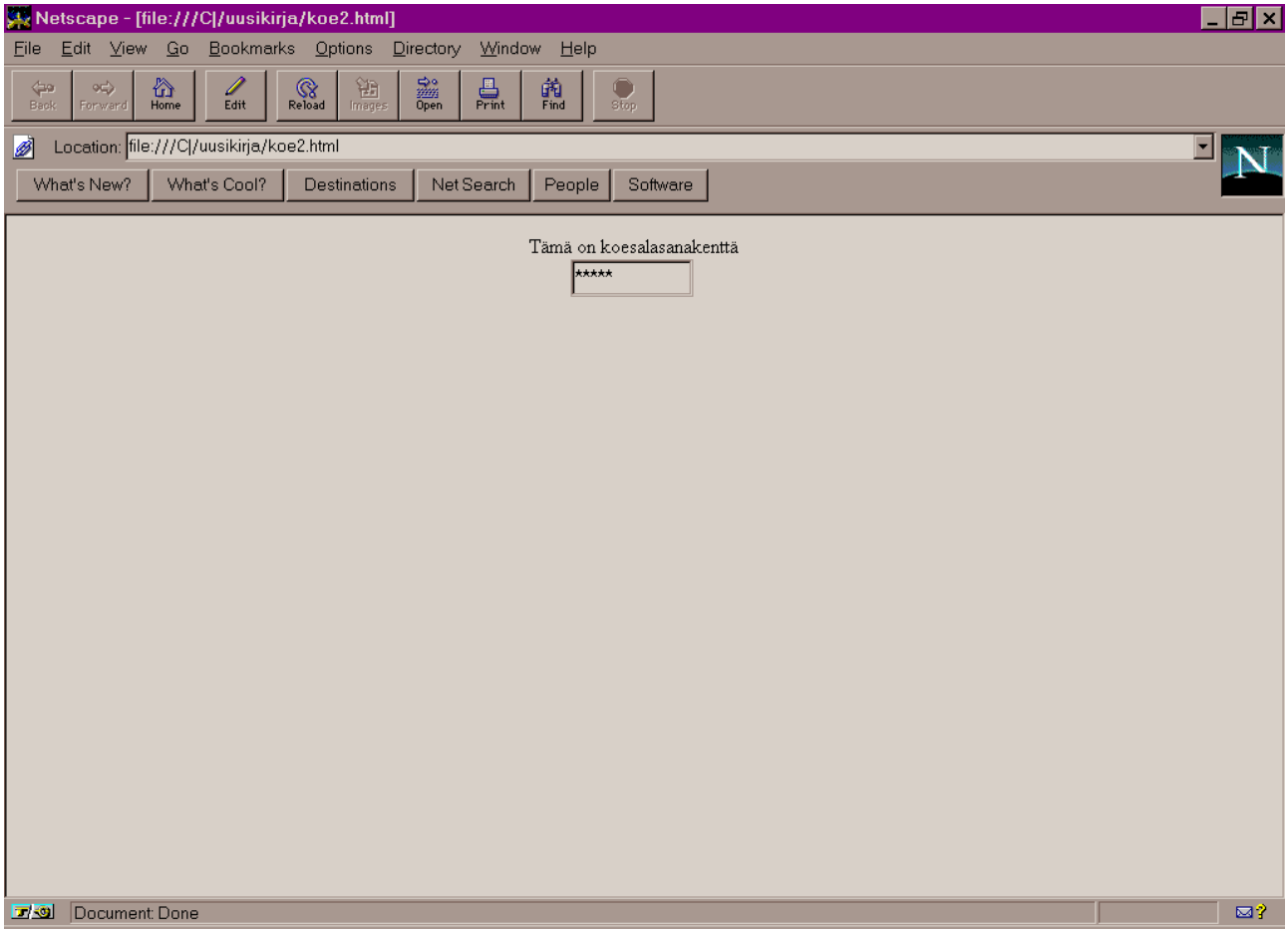
2.6.3 Salasanakenttä

Salasanakenttä on muuten normaali tekstikenttä, mutta siihen kirjoitetut merkit tulostuvat lukuohjelmassa tähtinä (*). Lomaketta käsittelevä ohjelma saa sen sijaan kentän sisällön selväkielisenä.

<INPUT type="*password*" [name="*nimi*"]
[maxlength="*maksimipituus*"] [size="*kentän koko*"]
[value="*oletus/alkuarvo*"]>

Esimerkki.

```
<HTML>
<BODY>
<FORM method="post"
action="http://www.kone.fi/cgi-bin/ohjelma">

<CENTER>Tämä on koesalasanakenttä<BR>
<INPUT type=”password” name=”eka” size=”10” va-
lue=”minä”>
</CENTER>
</FORM>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.6.4 Piilotetut kentät

Joskus on tarpeen lähettää lomakkeen mukana tietoa, jota ei ole syystä tai toisesta tarpeellista näyttää käyttäjälle ja jota käyttäjän ei tarvitse antaa. Tätä tarkoitusta varten on niinsanottu **hidden**-kenttä, jonka sisältöä ei voi muuttaa.

<INPUT type="*hidden*" [name="*nimi*"] value="*arvo*">

2.6.5 Painikkeet

Painikkeet ovat neliönmuotoisia (*checkbox*) tai pyöreitä (*radio*) kenttiä, joihin käyttäjä merkitsee ”ruksilla” valintansa.

<INPUT type=”*checkbox*” [checked] [name=”*nimi*”]
[value=”oletus*arvo*”]>

<INPUT type=”*radio*” [checked] [name=”*nimi*”] value=”oletus*arvo*”>

Jos **checked** on käytössä, painike on valmiiksi valittu.

Oletusarvo on tieto, jonka valittu painike palauttaa lomakkeen käsittelevälle ohjelmalle. Jos checkbox-painikkeelle ei anneta oletusarvoa, se palauttaa merkkijonon ”ON”, kun se on valittu.

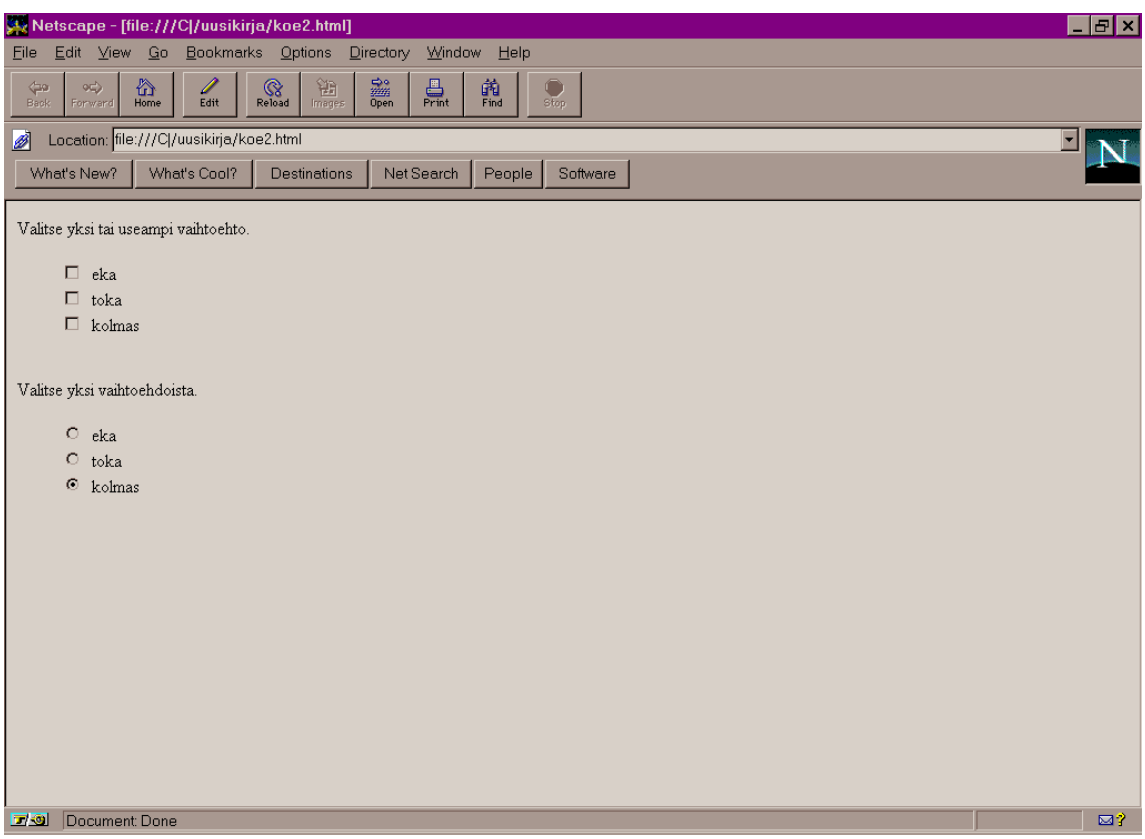
Jos painiketta ei ole valittu, se ei palauta mitään arvoa.

Radio-painiketta käytetään ryhmissä, joille annetaan kaikille sama nimi ja eri arvo.

Esimerkki.

```
<HTML>
<BODY>

<FORM method="post"
action="http://www.kone.fi/cgi-bin/ohjelma">

Valitse yksi tai useampi vaihtoehto.<BR>

<UL>

<INPUT type="checkbox" name="eka"> eka<BR>

<INPUT type="checkbox" name="toka"> toka<BR>

<INPUT type="checkbox" name="kolmas"> kolmas <BR>
</UL><BR>
```

```
        Valitse yksi vaihtoehdoista.<BR><UL>

        <INPUT type="radio" name="ryhma"  value="eka">
eka
        <BR>

        <INPUT type="radio" name="ryhma" value="toka">
toka
        <BR>

        <INPUT type="radio" name="ryhma" value="kolmas">
        kolmas

        </UL><BR>

        </FORM>
        </BODY>
        </HTML>
```



Edellisen esimerkin tuloste.

2.6.6 Valikot

Lomakkeeseen voidaan lisätä Windows-tyyppisiä valikoita, joista valitaan eteenpäin lähetettävä arvo.

<SELECT name="*nimi*" [multiple]>
<OPTION [selected] [value="*arvo*"]> *vaihtoehto*
…
</SELECT>

Vaihtoehto on valikkoon tälle kohdalle tuleva teksti.
Arvo on kyseisen valikon kohdan palauttama arvo, jos tätä ei ole määritelty, palautetaan *vaihtoehto* kohdalla oleva merkkijono.

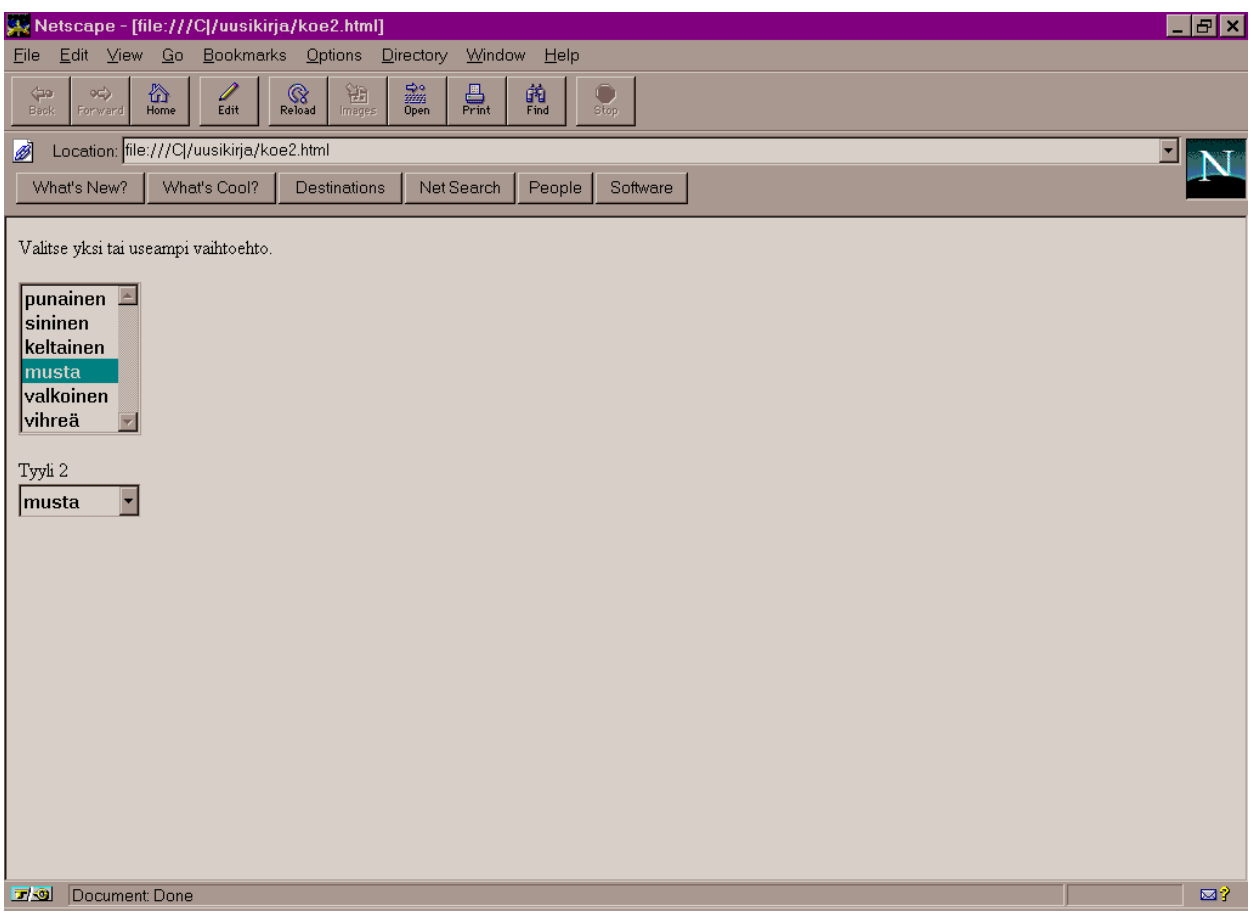
Jos **multiple** on otettu käyttöön, voidaan valikosta valita useampi kuin yksi kohta ja lukuohjelma myös näyttää koko valikon kerralla, muuten lukuohjelma näyttää vain yhden vaihtoehdon ja painamalla nuolenkuvaa valikossa saa muut vaihtoehdot tiputettua näkyville.

Kirjoittamalla **selected** jonkin valinnan kohdalle, saadaan tämä vaihtoehto maalattua oletusarvoksi.

Esimerkki.

```
<HTML>
<BODY>

<FORM method="post"
action="http://www.kone.fi/cgi-bin/ohjelma">

Valitse yksi tai useampi vaihtoehto.<BR><BR>

<SELECT name="valikko" multiple>

<OPTION> punainen
<OPTION> sininen
<OPTION> keltainen
<OPTION selected> musta
<OPTION> valkoinen
<OPTION> vihreä
```

```
</SELECT>
<BR><BR>Tyyli 2<BR>

<SELECT name="valikko2">

<OPTION> punainen
<OPTION> sininen
<OPTION> keltainen
<OPTION selected> musta
<OPTION> valkoinen
<OPTION> vihreä

</SELECT>

</FORM>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.6.7 Lähetyspainikkeet

Lähetyspainikkeilla saadaan lomakkeen sisältö lähettettyä verkon välityksellä sitä käsittelevälle ohjelmalle sekä lomakkeelle voidaan myös palauttaa oletusarvot tai tyhjentää kentät, jos oletusarvoja ei ole määritelty.

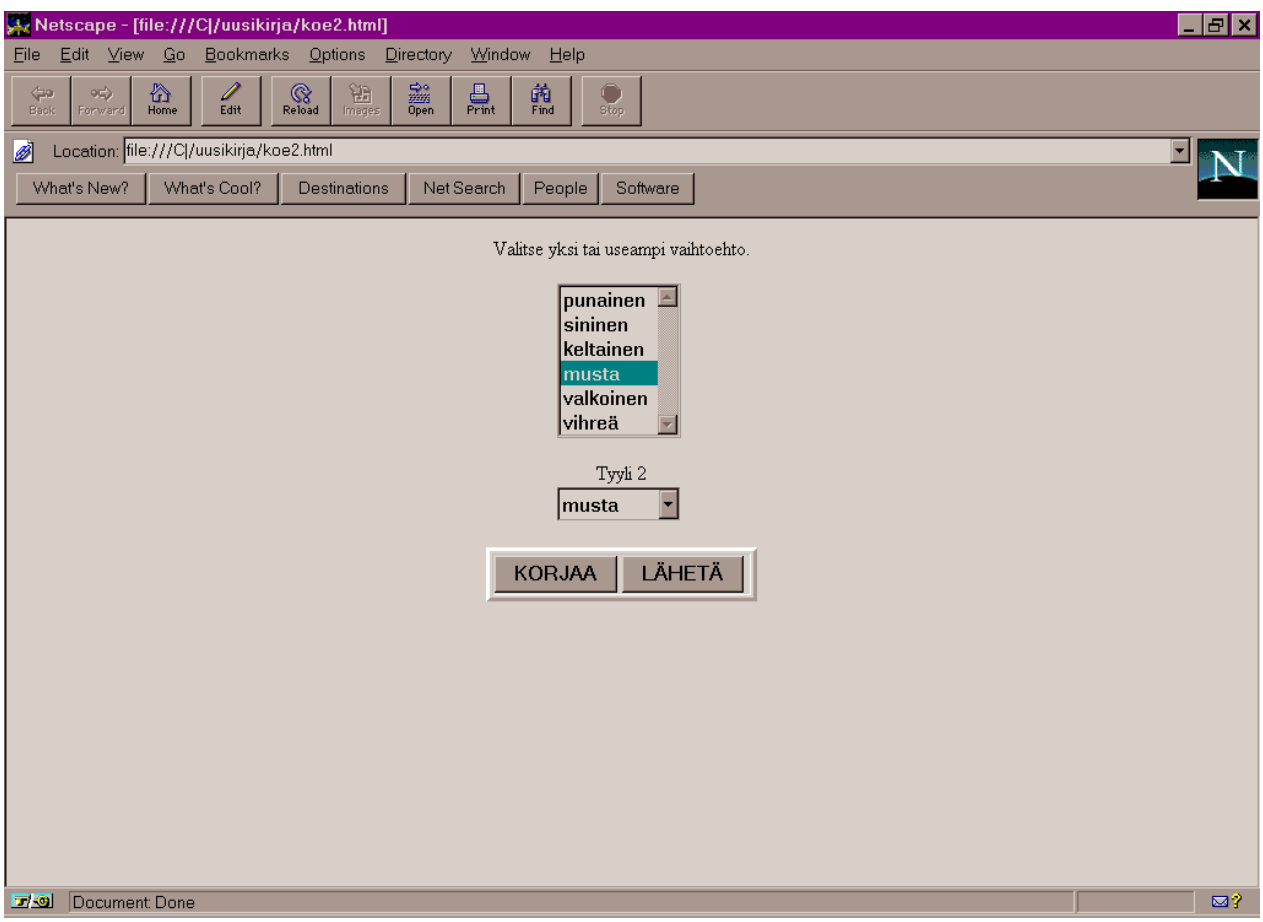
```
<INPUT type=”reset” value=”Palauta oletusarvot”>

<INPUT type=”submit”  value=”Lähetä lomake”>
```

Value-kenttään annettu arvo kirjoitetaan näkyviin painonappiin, jonka lukuohjelma tekee. **Reset** tyhjentää lomakkeen ja **submit** lähettää sen kun asianomaista painonappia on painettu.

Edellinen esimerkki lähetyspainikkeilla.

```
<HTML>
<BODY>
<CENTER>
<FORM method="post"
action="http://www.kone.fi/cgi-bin/ohjelma">

Valitse yksi tai useampi vaihtoehto.<BR><BR>

<SELECT name="valikko" multiple>

<OPTION> punainen
<OPTION> sininen
<OPTION> keltainen
<OPTION selected> musta
<OPTION> valkoinen
<OPTION> vihreä

</SELECT>
<BR><BR>Tyyli 2<BR>

<SELECT name="valikko2">
<OPTION> punainen
```

```
<OPTION> sininen
<OPTION> keltainen
<OPTION selected> musta
<OPTION> valkoinen
<OPTION> vihreä

</SELECT>
<BR><BR>
<TABLE border="3"><TR><TD>
<INPUT type="reset" value="KORJAA">
<INPUT type="submit" value="LÄHETÄ">
</TD></TR></TABLE>

</FORM>
</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

2.7 SSI-komennot

WWW-sivuille voidaan liittää myös suoraan palvelimessa ajettavien komentojen tulosteita niin sanottujen SSI-komentojen (*Server Side Include*) avulla edellyttäen, että tämä ominaisuus on palvelimen määrittelytiedostoissa otettu käyttöön. SSI-komentojen yleinen muoto on seuraava:

```
<!--#komento määrittely=”arvo” määrittely=”arvo” -->
```

SSI-komennot eivät näy lukijalle edes **View Source** -toiminnolla, vaan ainoastaan niiden aiheuttamat tulosteet näkyvät.

SSI-komentojen käyttöön ottamisessa on erilaisia tapoja ja paras onkin selvittää käytetty tapa palvelimen ylläpitäjältä. Palvelimeen saattaa olla esimerkiksi määritelty, että komentoja voidaan ajaa www-sivuilta, joiden tarkennin on *.shtml, tai jotkut yksittäiset komennot saattavat olla kokonaan estettyjä tai komentoja voidaan ajaa vain tiettyyn hakemistoon sijoitetuilta www-sivuilta.

2.7.1 Tiedoston lisäys

<!--#include [virtual=”*osoite*”] [file=”*tiedosto*”] -->

Osoite on suhteellinen osoite eli sama kuin URL, mutta ilman koneen nimeä.

Tällä komennolla voidaan lisätä dokumenttiin toisen tekstitiedoston sisältö. Haettava tiedosto voidaan määritellä joko osoitteella palvelimen dokumenttihakemistosta lähtien tai suoraan tiedoston nimellä. Tiedoston nimeä käytettäessä pitää tiedosto löytyä samasta hakemistosta kuin dokumentti, jolla sitä hakeva komento on.

2.7.2 Muuttujan tulostaminen

<!--#echo var="*muuttuja*" -->

Tällä komennolla voidaan tulostaa seuraavien muuttujien arvot dokumenttiin.

DOCUMENT_NAME	Senhetkisen dokumentin tiedostonnimi.
DOCUMENT_URL	Senhetkisen dokumentin sijainti palvelimessa.
QUERY_STRING_UNESCAPED	Dokumentille välitetyt parametrit.
DATE_LOCAL	Lisäyshetken kellonaika ja päivämäärä paikallisen aikavyöhykkeen mukaan.
DATE_GMT	Lisäyshetken kellonaika ja päivämäärä GMT (*Greenwich Median Time*) -aikavyöhykkeen mukaan.
LAST_MODIFIED	Dokumentin viimeisin muutosaika.

2.7.3 Muiden dokumettien tiedot

<!--#fsize [virtual="*osoite*"] [file="*tiedosto*"] -->

<!--#flastmod [virtual="*osoite*"] [file="*tiedosto*"] -->

Tiedoston haku kuten komennossa **include** (tiedoston lisäys).

Fsize tulostaa määritellyn tiedoston koon ja **flastmod** tulostaa määritellyn tiedoston viimeisen muutospäivämäärän.

2.7.4 Komentojen ja ohjelmien tulosteet

<!--#exec [cmd="*komento*"] [cgi="/cgi-bin/*CGI-ohjelma*"] -->

Komento suoritetaan palvelimessa **/bin/sh**:ssa. CGI-ohjelman paikalle ajettavan cgi-ohjelman nimi, jonka on sijaittava palvelimen vastaavassa hakemistossa. Nämä komennot saattavat olla estetettyjä useissa palvelimissa.

Esimerkki. Päiväyksen tulostaminen **date**-komennolla.

```
<HTML>
<BODY>
Tänään on
<!--#exec cmd="date " -->
</BODY>
</HTML>
```

2.7.5 Muotoilukäskyt

<!--#config [errmsg="*virheilmoitus*"] [timefmt="*aikaformaatti*"] [sizefmt="*kokoformaatti*"] -->

Virheilmoitus on teksti, joka korvaa vakiovirheilmoituksen jos SSI-komentojen suorituksessa tapahtuu virheitä.

Aikaformaatti määrää ajan esitystavan. Katso (unix-koneessa) *man strftime.*

Kokoformaatti määrää tiedostojen koon esittämistavan. Mahdollisia arvoja on kaksi. **Bytes**, joka kertoo, että koko ilmaistaan tavuina. **Abbrev**, joka kertoo, että koko pyöristetään lähimpään kilo- tai megatavuun.

3. JavaScript-kielen perusteet

Tässä luvussa perehdytään JavaScript-kieleen.

3.1 JavaScript ja Java

JavaScript-kieli muistuttaa Java-kieltä koodiltaan, mutta se ei ole niin vahvasti tyypitetty ohjelmointikieli kuten Java tai C++. JavaScript tukee useimpia Javan lauserakenteita. JavaScript-koodia ei tarvitse erikseen kääntää kuten Java-koodia, vaan koodi liitetään suoraan HTML-dokumenttiin. JavaScript- ja Java-kieltä voidaan käyttää yhdessä samassa HTML-dokumentissa. Edellä mainituista syistä JavaScript-koodin kirjoittaminen sujuu aloittelijalta joutuisammin kuin Java-koodin. Myös JavaScript vaatii vähintään Netscape 2.0 tai uudemman lukuohjelman toimiakseen.

Oheiseen taulukkoon on koottu JavaScript- ja Java kielen eroavaisuuksia.

Java	*JavaScript*
Käännetään ennen suoritusta.	Tulkataan suorituksen aikana.
Perustuu luokkiin, jotka ovat periytyviä ja aktivoituessaan synnyttävät olioita. Ei sisään rakennettuja olioita.	Olio-pohjainen, ei periytyvyyttä eikä luokkia, sisään rakennettu laaja olio valikoima.
Ladataan erillisestä tiedostosta HTML-dokumenttiin.	Liitetään suoraan HTML-dokumenttiin.
Vahva muuttujien tyypitys ja esittely.	Ei muuttujien tyypitystä eikä esittelyä.
Staattinen: Viittaukset olioihin huomioidaan käännösvaiheessa.	Dynaaminen: Viittaukset olioihin huomioidaan vasta suorituksen aikana.
Ei voi kirjoittaa kiintolevylle.	Ei voi kirjoittaa kovalevylle.

3.2 Kielen rakenne ja kirjoitusasu

JavaScript-kielinen ohjelma koostuu sisään rakennetuista valmiista olioista, joilla on useita metodeita (olion funktio) ja erilaisia muuttujia sekä funktioista ja olioista, jotka voidaan määritellä koodissa. Funktiot kuten kaikki koodi JavaScript-kielessä määritellään <SCRIPT> … </SCRIPT> tagien sisään suoraan HTML-dokumenttiin. Käytettäessä JavaScriptiä HTML-dokumentissa, se vuorottelee usein standardin HTML:n kanssa, jolloin yksittäiset toiminnot voidaan toteuttaa sillä kielellä, joka parhaiten sopii asianomaiseen tarkoitukseen. HTML-koodiin voidaan kirjoittaa kommentti <!-- … -- > tagi, joka estää <SCRIPT> … </SCRIPT> tagien ja niiden välissä olevan koodin näkymisen vanhemmissa netscape-versioissa.

```
<HTML>
<HEAD>
<!--  Piilota tämä osa vanhoilta browsereilta -- >

<SCRIPT LANGUAGE=”JavaScript”>
        //   JavaScript-koodia
</SCRIPT>

<!--  Lopeta piilottaminen  -- >

</HEAD>
<BODY>
<BODY bgcolor=”#ffffff”>
<!--   html-koodia  -- >

<SCRIPT LANGUAGE=”JavaScript”>
       // JavaScript-koodia
</SCRIPT>

<!-- html-koodia  -- >
</BODY>
</HTML>
```

Java-kielen tavoin myös JavaScript sisältää lauseita ja funktiokutsuja. Ohjelman suoritettava osa koostuu lauseista, jotka sisältävät yhden tai useamman lausekkeen. Lauseke on puolestaan muuttuja, operaattori tai

funktiokutsu, jolla on jokin arvo. Seuraavassa on esimerkkejä yksinkertaisista lausekkeista.

```
2+3
laske()
```

Lauseella ei ole arvoa ja se on hyvä päättää puolipisteeseen, vaikka se ei aina ole tarpeen. Lause voi olla esimerkiksi seuraavanlainen.

```
tulos = 2 + 3;
```

JavaScriptissä käytetään aaltosulkeita sitomaan useita lauseita yhteen. Esimerkiksi seuraavassa for-lauseen vaikutusaluetta kasvatetaan aaltosulkeilla osoittamaan myös toista lauseketta, kun se normaalisti vaikuttaa vain sitä välittömästi seuraavaan lauseeseen.

```
for (var k = 0 ; k < 10 ; k++)
{
  tulos = 2 + k ;

  document.write("tulos : ", tulos):
}
```

3.3 Kommentit

JavaScript tukee samoja kommentteja kuin Java ja C++ -kieli.

```
// Tämä on kommentti

/*  Tämä on
    Toinen kommentti  */
```

3.4 Muuttujat ja tietotyypit

JavaScript tuntee seuraavat muuttujatyypit:

numerot:

esim. 39 tai 2.6783

loogiset eli Boolen totuusarvot:

true (tosi) tai **false** (epätosi)

merkkijonot:

esim. “Tämä on merkkijono”

null:

varattu sana, joka ilmaisee nolla-arvoa.

Taulukkomuuttujia eli niin sanottuja indeksoituja muuttujia ei voida suoraan määritellä JavaScriptiin, vaan ne on erikseen luotava. Taulukkomuuttujiin palataan myöhemmin. Varattuja sanoja ei voi käyttää muuttujien niminä. Seuraavassa on sallittuja muuttajan nimiä.

```
ekamuuttuja
Muuttuja_1
_eka
```

eli muuttujanimissä voi esiintyä kirjaimet väliltä a-z ja A-Z, numerot ja alaviiva (_). Muuttujanimi on aloitettava joko kirjaimella tai alaviivalla. JavaScript tunnistaa isot ja pienet kirjaimet eri merkeiksi. Esimerkiksi.

MuuTTuja ja muuttUJA -nimillä tarkoitetaan eri muuttujia.

Muuttuja, johon ei ole sijoitettu mitään arvoa, aiheuttaa ajonaikaisen virheilmoituksen. Esimerkiksi muuttuja tulos aiheuttaisi tällaisessa tapauksessa ajonaikaisen virheilmoituksen:

```
tulos ;

tulos is not defined.
```

Tämä voidaan välttää sijoittamalla varattu sana **var** muuttujanimen eteen:

```
var tulos ;
```

Varatut sanat:

abstract	**do**	**if**	**package**	**throw**
boolean	**double**	**implements**	**private**	**throws**
break	**else**	**import**	**protected**	**transient**
byte	**extends**	**in**	**public**	**true**
case	**false**	**instanceof**	**return**	**try**
catch	**final**	**int**	**short**	**var**
char	**finally**	**interface**	**static**	**void**
class	**float**	**long**	**super**	**while**
const	**for**	**native**	**switch**	**with**
continue	**function**	**new**	**synchronized**	
default	**goto**	**null**	**this**	

Tietotyyppejä ei ohjelman tekijän tarvitse itse määritellä, koska JavaScript huolehtii tyyppien muunnoksista noudattaen seuraavaa käy-

täntöä tulkiten lauseita vasemmalta oikealle. Muunnostyyppi määräytyy vasemman puoleisen operandin mukaisesti.

1.

```
var   MERKKIJONO =  “12” ;

var  LUKU = 3 ;

x = LUKU + MERKKIJONO  ;
```

Tässä x saa arvokseen luvun 15 .

2.

```
var MERKKIJONO = “12” ;

var LUKU = 3 ;

x = MERKKIJONO + LUKU  ;
```

Tässä x saa arvokseen merkkijonon “123” .

Esimerkissä 1 muuttujaan **LUKU**, johon on sijoitettu luku 3 lisätään muuttujan **MERKKIJONO** sisältämä arvo (“12”). Täten x saa arvokseen 15, koska tyypin muunnos suoritetaan muuttujan **LUKU** tyypin mukaan, joka on luku. Eli merkkijono “12” muutetaan luvuksi 12 ja suoritetaan laskutoimitus 3 + 12 = 15.

Esimerkissä 2 muuttujaan **MERKKIJONO**, johon on sijoitettu merkkijono “12” lisätään muuttujan **LUKU** arvo (luku 3) ja x saa täten arvokseen merkkijonon “123”, koska tyypin muunnos suoritetaan muuttujan

MERKKIJONO tyypin mukaan, joka on merkkijono. Eli luku 12 muutetaan merkkijonoksi “12” ja suoritetaan kahden merkkijonon yhdistäminen.

Edellisissä esimerkeissä täytyy muuttujan **MERKKIJONO** sisältö olla jokin numero merkki (‘1234567890’) tai muuten tyypin muunnos kyseistä toimitusta varten ei onnistu, vaan tuottaa virheilmoituksen.

Seuraavassa taulukossa on yhteenveto tyypinmuunnoksen tuloksesta eri tyyppien välillä kun vasemman puoleisen sarakkeen tietotyyppi muunnetaan toiseksi ylimmän rivin tietotyypiksi.

	Muunnetaan tyypiksi				
tieto tyyppi	**funktio**	**olio**	**luku**	**Boole**	**merkki jono**
funktio	-	funktio	error	error	muodostetaan
olio	error	-	error	true	merkkijono
NULL olio	funktio = olio ->OK	-	0	false	“null”
luku (!=0)	error	luku	-	true	merkkijono
0	error	NULL	-	false	“0”
Error (Nan)	error	luku	-	false	NaN
+ ääretön- myys	error	luku	-	true	‘+infiniy’
- ääretön- myys	error	luku	-	true	“-infinity”
Boole:					
false	error	Boole	0	-	“false”
true	error	Boole	1	-	“true”
merkki jono (ei NULL)	jos merkkijono funktion	merkki jono	jos merkki jonossa nume-	true	-

	nimi -> OK		roita -> OK		
NULL - merkki-jono	error	merkki-jono	jos merkki-jonossa “0” merkki -> OK	false	-

Luvut ja lukujärjestelmät esitetään kuten Javassa. Liukuluku koostuu kokonaisluvusta, desimaalipisteestä ja desimaaliosasta sekä eksponentistä. Liukuluku voidaan pakottaa ekspotentiaaliseksi, jos luvun keskellä esiintyy kirjain E tai e. Seuraavassa on esimerkkejä liukuluvuista.

3.14
3.14E12 = 3.14 * 10^12

Lukujärjestelmiä on käytettävissä kolme: desimaali, heksadesimaali ja oktaali. Ne merkitään seuraavasti:

Desimaaliluku on puhdas luku (ei saa alkaa nollalla)
- Kantaluku 10
- **Esimerkki**: 100

Heksadesimaaliluku alkaa 0x-yhdistelmällä
- Kantaluku 16
- **Esimerkki**: 0x100

Oktaaliluku alkaa nollalla
- Kantaluku 8
- **Esimerkki**: 010

3.5 Merkkijonot

Merkkijono sijoitetaan joko “ “-merkkien tai ‘ ‘-merkkien sisään. JavaScriptissä on käytössä samat operaattorit sekä tulosteen muotoilu merkit kuin Javassa. Huomaa, että taulukoita ei voi suoraan määritellä. Merkkijonojen käsittelyyn palataan myöhemmin String-olion yhteydessä.

“nimi”
‘arvo’
“A”
‘2’

\n uusi rivi
\t tabulaattori, horisontaalinen
\b kohdistin siirtyy yhden merkin vasemmalle

huom ! Muotoiltava teksti on sijoitettava tagien <PRE> …</PRE> sisään

Esimerkki:

```
document.write(“<PRE> Tämä\t on\n koe </PRE>”);
```

Tuottaa tulosteen:

Tämä on
koe

3.6 Operaattorit

Tässä kappaleessa on pääosin kerrattu se, mitä on aikaisemmin todettu Javan yhteydessä.

Operaattoreilla on kielessä suuri merkitys. Operaattorien avulla tehdään laskutoimituksia, verrataan lukuja toisiinsa ja sijoitetaan muuttujiin lukuja. Myös operaattoreita voidaan yhdistää toisiinsa, jolloin saadaan hyödyllisiä arvonmuunto-operaattoreita.

Sijoitusoperaattori on tuttu yhtäsuuruusmerkki **=**. Sitä ei saa sekoittaa vertailuoperaattoriin ==, koska sillä sijoitetaan muuttujaan sen arvo. Vertailuoperaattorilla puolestaan vertaillaan arvojen yhtäsuuruutta. Ohjelmointivirhe saattaa syntyä **if**-lauseessa, kun == -operaattorin tilalle laitetaan erehdyksessä = -operaattori.

sijoitus:

```
var tulos = 2
```

eli muuttujan **tulos** arvoksi sijoitetaan luku 2.

vertailu:

```
if( tulos == joku_muu)
```

eli muuttujan **tulos** arvoa verrataan muuttujan **joku_muu** arvoon.

Aritmeettiset operaattorit on tarkoitettu lukujen laskutoimituksiin. Kielessä on valmiina vain peruslaskutoimitukset, mutta esimerkiksi neliöjuuri ja potenssiinkorotus suoritetaan **Math**-olion metodeilla.

+	yhteenlasku
-	vähennyslasku
*	kertolasku
/	jakolasku
%	jakojäännös

Merkkijonojen yhteydessä + -operaattorilla voidaan liittää merkkijonoja toisiinsa. Esimerkiksi

```
var eka = “jono” ;
var toka = “merkki” ;
var tulos = toka + eka ;                    // liitä merkkijonot

document.write(tulos);                      //tulosta teksti
```

Tuloste.

```
merkkijono
```

Aritmeettiset operaattorit ja sijoitusoperaattorit voidaan myös yhdistää seuraavasti:

```
var eka = 2 ;
var toka = 3 ;
var toka += eka ;

document.write(toka);              // tulostaa luvun 5
```

Eli edellinen vastaa rakennetta:

```
var toka = toka + eka ;
```

Saman voi toteuttaa myös merkkijonoilla.

```
var eka = “jono” ;
var  toka = “merkki” ;
var toka += eka ;                    // toka = toka + eka

document.write(toka);                //tulostateksti
```

Tuloste.

merkkijono

Arvonmuunto-operaattorit ovat tarkoitettu yhden muuttujan arvon muuntamiseen. Lukuun on mahdollista lisätä/vähentää yksi, lisätä/vähentää toinen luku tai tehdä luvusta negaatio (vaihtaa luvun etumerkki). Myös kerto- ja jakolaskutoimitukset ovat mahdollisia.

++	lisää yksi
--	vähennä yksi
~	kahden komplementti
-	negaatio
+=	lisää luku
-=	vähennä luku
*=	kerro luku
/=	jaa luku

Vertailuoperaattoreita tarvitaan silloin, kun lukuja pitää verrata toisiinsa. Operaattorit ottavat huomioon luvun *etumerkin*, joten negatiivinen luku on aina pienempi kuin positiivinen. Mikäli halutaan verrata lukuja itseisarvojen perusteella, pitää käyttää **Math**-olion **abs**-metodia.

==	yhtäsuuri
!=	erisuuruus
>	suurempi kuin
<	pienempi kuin

<=	pienempi tai yhtäsuuri kuin
>=	suurempi tai yhtäsuuri kuin

Loogiset operaattorit ovat hyviä silloin, kun tarvitaan ehtoja lukujen välillä ennen kuin koko lauseke asetetaan todeksi.

!	EI -operaatio
&&	JA -operaatio
||	TAI -operaatio

Esimerkiksi, jos muuttujien **a** ja **b** tulee olla arvoiltaan 5 ja 10, niin käytetään **JA** -operaatiota seuraavalla tavalla.

```
( a == 5  &&  b == 10)
```

Bittitason loogiset operaattorit toimivat nimensä mukaisesti bittitasolla. Operaattoreita ei käytetä ehtojen asettamiseen, kuten yllä olevia loogisia operaattoreita.

!	EI -operaatio
&	JA -operaatio kahden luvun välillä
|	TAI -operaatio kahden luvun välillä
^	XOR -operaatio kahden luvun välillä

Katso bittioperaattorien käytöstä vastaavasta kappaleesta Javan yhteydestä. Muita bittitason käsittelyyn liittyviä operaattoreita ovat bittien siirto-operaattorit. Ne nimensä mukaisesti siirtävät luvun bittejä vasemmalle tai oikealle. Tällä tavoin voidaan suorittaa kerto- ja jakolasku kahden potensseilla (1,2,4,8 jne).

<<	Kierrätä vasemmalle
>>	Kierrätä oikealle (kopioi etumerkkiä)

>>>	Kierrätä oikealle (etumerkki positiiviseksi)
<<=	Kierrätä vasemmalle ja sijoita
>>=	Kierrätä oikealle ja sijoita (kopioi etumerkkiä)
>>>=	Kierrätä oikealle ja sijoita (etumerkki positiiviseksi)

Operaattorien suoritusjärjestys on seuraava ellei sitä sulkeiden avulla muuteta.

Pilkku (,)
= += -= *= /= %= <<= >>= >>>= &= ^= |=
conditional ?:
||
&&
|
^
&
== !=
< <= > >=
<< >> >>>
+ -
* / %
! ~ - ++ --
funktiokutsu() [] .

3.7 Kontrollilauseet

Ehtolause **if-else** on lause, joka testaa onko annettu ehto tosi. Jos ehto on tosi suoritetaan if-haaran lauseke, muulloin se jätetään suorittamatta. Ehtolausekkeeseen voidaan liittää esimerkiksi mikä tahansa vertailu-operaattori.

```
if (ehto)
   lause;

if (muuttuja == 3)
    lause;
```

If-haaraan voidaan liittää myös **else**-haara. Mikäli annettu ehto ei toteudu, suoritetaan **else**-haaraan määräämä lauseke. Esimerkiksi seuraavassa testataan onko muuttuja **A** tosi vai epätosi, ja asetetaan muuttuja **B** sen mukaan.

```
boolean a=1;
boolean b;

        if (a==TRUE)
         b=TRUE;

        else
         b=FALSE;
```

Toisen tyyppinen ehtolauserakenne saadaan aikaan lauseella, joka on seuraavaa muotoa.

```
muuttuja = (ehto)?eka : toka
```

Jos ehto on tosi, muuttujaan sijoitetaan *eka*n arvo ja jos ehto on epätosi muuttujaan sijoitetaan *toka*n arvo. Esimerkiksi:

```
etumerkki = (luku < 0)?”negatiivinen” : “positiivinen” ;
```

Muuttujan **etumerkki** sijoitetaan merkkijono “negatiivinen”, jos ehto (muuttujan **lukuarvo** on määritelty koodissa aikaisemmin) **luku** < 0 on tosi. Muussa tapauksessa muuttujan **etumerkki** sijoitetaan merkkijono “positiivinen”.

Toistolauseita on olemassa kaksi. Ne ovat nimeltään **for** ja **while**, joista **for**-lausetta on lisäksi olemassa kahta eri tyyppiä. Ensimmäistä **for**-lausetta käytetään, kun tiedetään kuinka monta kertaa silmukka suoritetaan. Toinen **for**-lausetyyppi muistuttaa ulkoasultaan *unix*:n komentotiedostoissa käytettävää tyyppiä. **While**-lausetta käytetään sellaisissa tilanteissa, joissa silmukoita suoritetaan niin kauan kuin ehto toteutuu. Lukumäärää ei tarvitse tuntea etukäteen.

For-toistolause muodostuu kolmesta osasta. Ensin alustetaan laskuri, sen jälkeen on toistoehto, ja kolmanneksi lisätään laskuriin askeleen osoittama lukumäärä for-haaran lauseita suoritetaan niin monta kertaa kuin ehto on voimassa. Esimerkiksi seuraava silmukka suoritetaan 10 kertaa.

```
for ( var i=0; i<=9; i++)
  summa+=i;
```

Toinen **for**-lause on ulkoasultaan ja toiminnaltaan edellisestä täysin poikkeava. Siinä ensiksi mainittu muuttuja saa varattua sanaa **in** seuraavan olion muuttujat vuorotellen arvokseen ja suoritus päättyy kun muuttujat loppuvat. Olion sisältämiin ominaisuuksiin palataan myöhemmin. Seuraavassa esimerkissä olion *koeolio*-tyypille on määritelty kolme muuttujaa, joiden nimet ovat ; *eka*, *toka* ja *kolmas*. Muuttujiin on lisäksi sijoitettu luvut 1, 2 ja 3. Muuttujien nimet voidaan tulostaa ja niiden sisältöä osoittaa seuraavasti

```
for(var k in koeolio)
{
  document.write("Muuttuja : “ + koeolio + “<BR>Arvo =  “ +
   koeolio[k]  + “<BR><BR>”); }
```

Esimerkki tuottaa tulosteen:

```
Muuttuja : eka
Arvo =  1

Muuttuja : toka
Arvo =  2

Muuttuja : kolmas
Arvo =  3
```

While-toistolause muodostuu while-käskystä, ja sitä seuraavasta lauseesta. Lausetta toistetaan niin kauan kuin ehto on tosi. Esimerkiksi seuraavassa suoritetaan lause summa+=i 10 kertaa.

```
while (++i<10)
summa+=i;
```

Jos do-haara liitetään while-lauseeseen saadaan **do-while-toistolause**. Do-haara suoritetaan se vähintään kerran vaikka while-haaran ehto ei toteutuisi.

```
do
  lause;
while (ehto);
```

Lisäksi while-lauseessa voi olla ehdottomia hyppykäskyjä (**break** ja **continue**), joiden avulla silmukan suorittaminen voidaan pysäyttää tai aloittaa alusta. Seuraava esimerkki lisää k:n arvoa yhdellä, jos ehto on tosi ja lopettaa lisäämisen kun k saavuttaa arvon 100.

```
while (ehto)
{   k++;
    if (k>100)
      break
}
```

With-lauseella voidaan oliosta tehdä oletusolio lauseessa esiintyville muuttujille. Lause on seuraavaa muotoa.

```
with (olio)
{
  lauseet
}
```

Muuttujat, jotka esiintyvät ilman olioviittausta, oletetaan kuuluvan sulkeiden sisällä olevaan olioon. Esimerkiksi

```
with (Math)
{
  a = PI * r*r ;
  x = r * cos(alfa);
  y = r * sin(alfa);
}
```

vastaa lauseita:

```
a = Math.PI * r*r ;
x = r * Math.cos(alfa) ;
y = r * Math.sin(alfa) ;
```

3.8 JavaScript-kielen olioista

Toimiva JavaScript-koodi koostuu olioista. Olion rakenteeseen sisältyy muuttujia ja metodeita. Metodilla kuvataan funktiota, joka kuuluu jonkin olion rakenteeseen. Olion muuttujiin voidaan sijoittaa myös toisia olioita ja taas näiden olioiden muuttujiin uusia olioita jne. Olion muuttujaan viitataan rakenteella:

```
olion_nimi.muuttujan_nimi
```

Esimerkki:

```
button.value
```

Olion muuttujaan sijoitetaan arvo seuraavasti. Oletetaan, että olio on luotu aikaisemmin.

```
toimi.value = “laske” ;
```

tai

```
tulos.arvo = toimi ;
```

Olion muuttujan arvoon päästään käsiksi seuraavasti.

```
var tulos = toimi.value ;
document.write(tulos);
```

tai

```
document.write(toimi.value);
```

Molemmat tuottavat saman tulosteen (arvona yllä sijoitettu “laske”):
jos olion muuttujaan on sijoitettu toinen olio, viitataan tämän olion muuttujaan seuraavasti

```
var valmis = tulos.arvo.value ;
```

Eli olion *tulos* muuttujaan *arvo* on sijoitettu olio *toimi*, jonka muuttujaan *value* on sijoitettu merkkijono “laske”, joka sijoitetaan muuttujaan *valmis*.

```
document.write(“Suoritettiin käsky : “,tulos.arvo.value);

tai

document.write(“Suoritettiin käsky : “,valmis);
```

Kummatkin tuottavat saman tulosteen:

```
Suoritettiin käsky :  laske
```

Olioiden muuttujia voidaan osoittaa myös toisella tavalla, joka muistuttaa muista kielistä tuttua taulukkorakennetta.

```
toimi[“value”]
```

vastaa rakennetta

```
toimi.value
```

Muuttujan nimi **value** voidaan korvata sitä vastaavalla indeksillä eli järjestysluvulla, jonka avulla ilmaistaan sen sijoittuminen olion muuttujien määrittelylistassa. Huomaa, että ensimmäistä muuttujaa osoitetaan indeksillä 0.

Esimerkki: Oletetaan, että oliolle *toimi* on määritelty kaksi muuttujaa nimiltään **value** ja **name** tässä järjestyksessä ja joihin on sijoitettu merkkijonot “laske” ja “yhdistä”.

```
Document.write(toimi[0],”<BR>”,toimi[1]) ;
```

tuottaa tulosteen:

laske
yhdistä

3.9 Funktiot ja metodit

Funktioksi kutsutaan olion muodostuksen perusrakennetta ja sen toiminta määritellään aikaisemmin kuvatuilla lauseilla, jotka on rajattu { …} sulkeiden sisään.

Metodi on funktio, joka kuuluu johonkin olioon. Funktion määrittely aloitetaan varatulla sanalla **function** ja lisäämällä sen perään funktion nimi ja mahdolliset parametrit aaltosulkeissa.

Esimerkki: Yksinkertainen funktio, joka tulostaa näytölle halutun tekstin.

```
function kirjoita( ) {

    document.write(“Tämä on koe”);  }
```

Funktion määrittely ei aiheuta sen suorittamista, vaan suoritus tapahtuu vasta, kun funktiota kutsutaan jossain saman HTML-dokumentin alueella sen nimellä ja samalla välitetään mahdolliset parametrit. Funktio on määriteltävä koodissa ennen kuin sitä kutsutaan ja näin ollen on hyvä sijoittaa kaikki ohjelman funktiot <HEAD> … </HEAD> tagien sisään, jolloin ne ladataan muistiin ensimmäisenä. Yllä mainitun funktion suoritus koodissa tapahtuu käskyllä:

```
kirjoita();
```

jolloin näytölle tulostuu:

Tämä on koe

Funktioissa toimii niin sanottu parametrin välitys eli funktion kutsun yhteydessä välittyy kutsuttavalle funktiolle parametreina olevien muuttujien arvot, jotka funktio saa argumenteikseen. Kutsussa olevien muuttujien arvot eivät muutu funktion suorituksen aikana. Vastaan ottavan funktion muuttujien ja parametrien ei tarvitse olla saman nimisiä kuin kutsun parametreina toimivat muuttujat. Parametriksi voidaan sijoittaa muuttuja, puhdas luku ja merkkijono sekä olio tai toisen funktion kutsu. Funktiossa määritellyt muuttujat ovat voimassa ainoastaan funktion sisällä. Funktioihin liittyy myös niin sanottu paluu arvo joka määrää sen mitä funktio palauttaa kun sitä kutsutaan. Palautus käsky on nimeltään **return**. Funktion paluuarvo palautetaan sen nimen kautta.

Esimerkki: *apu* niminen muuttuja saa *koefunktio* nimisen funktion paluuarvon arvokseen.

```
apu = koefunktio(par1, par2);
```

Seuraavassa esimerkissä määritellään funktio **laskek_arvo**, jonka parametreina ovat muuttujat **eka**, **toka** ja **kolmas**. Funktio saa argumenteikseen parametrien kautta välitetyt luvut ja laskee lukujen keskiarvon ja palauttaa sen palautusarvonaan. Molemmat määrittelytavat johtavat samaan lopputulokseen, mutta ensimmäinen on havainnollisempi tarkasteltaessa koodia joskus myöhemmin. Tulostus näytölle hoidetaan **document.write** -metodin avulla, jonka saa argumentikseen **laskek_arvo()** funktiokutsun tuottaman funktion **laskek_arvo** palautusarvon.

```
<SCRIPT>

function laskek_arvo(eka, toka, kolmas)
{
   k_arvo = (eka+toka+kolmas)/3 ;
   return k_arvo ;
}

/*     function laskek_arvo(eka, toka, kolmas)
```

```
        {
          return  (eka+toka+kolmas)/3 ;
        }

     HUOM! Samannimisten funktioiden määrittelyjä ei voi
    käyttää koodissa yhtäaikaa
    */


    var luku_1 = 12 ;         // sijoitetaan arvot muuttu-
    jiin
    var luku_2 = 4
    var luku_3 = 5 ;


    document.write(“Keskiarvo = “laskek_arvo(luku_1, lu-
    ku_2,
      luku_3));
    </SCRIPT>
```

Yllä oleva ohjelma tuottaa seuraavan tulosteen.

Keskiarvo = 7

Seuraavassa esimerkissä funktiota **potenssi** käytetään funktion **laske** parametrina sen kutsun yhteydessä ja funktio **laske** saa argumentikseen funktion **potenssi** palautusarvon. Lisäksi funktiota **potenssi** käytetään funktion **laske** sisäisessä lauseessa.

```
    <HTML>
    <HEAD>
    <SCRIPT>

// Funktiot ladataan muistiin ensimmäisinä …

    function potenssi (luku, exponentti)
    {
```

```
        if( exponentti == 0)     // jos exponentti = 0,
palauteetaan 1
      { return 1 ;}

     if( exponentti == 1)     // jos exponentti = 1, pa-
    lautetaan luku
      { return luku ;}

     var apu = luku ;  // otetaan muistiin alkuperäinen
    kantaluku

     for (k=0 ; k < exponentti -1 ; k++)
     {
       luku = apu * luku ;    // potenssiin korotus
      }
     return luku ;
    }

    function laske(luku_a, luku_b, luku_c, luku_d)
    {
      /* Funktiossa lasketaan kaava (x3- y4) /( zx +t)
    käyttäen apuna   aikaisemmin määriteltyä potenssiin-
    korotus funktiota. Kaava on keksitty eikä esitä mi-
    tään matemaattista tai fysikaalista riippuvuutta. */


     tulos = (potenssi(luku_a, 3) - potenssi(luku_b, 4))
      /  (potenssi(luku_c, luku_a) + luku_d) ;
      return tulos ;
    }

    /*… kun ne on sijoitettu <script> … </script> tagien
    sisään,
    jotka on sijoitettu <head> … </head> tagien sisään.
    */

    </SCRIPT>
    </HEAD>
    <BODY>
    <SCRIPT>

        var eka = 2 ;
        var toka = 3 ;
        var kolmas = 4 ;
        var neljäs = 5
```

```
        document.write("Tulos = ", laske(eka, toka, kol-
        mas,
          potenssi(neljäs, 4)));

    </SCRIPT>
    </BODY>
    </HTML>
```

Ohjelma tulostaa näyttöön :

Tulos = -0.11388455538221529

Metodi luodaan määrittelemällä ensin funktio yllä kuvatulla tavalla ja sijoittamalla se olioon. Esimerkki: Olioon **toimi** sijoitetaan metodiksi funktio **potenssi** ja metodin nimeksi tulee myös **potenssi.**

```
toimi.potenssi = potenssi ;
```

Syntynyttä metodia kutsutaan seuraavasti ja kutsulla suoritetaan funktio **potenssi** annetuilla parametreillä.

```
toimi.potenssi(parametri1, parametri2) ;
```

3.9.1 This-osoitin

JavaScript tuntee erityisen **this**-osoitinrakenteen, jolla pystytään syöttämään olio itsensä argumenttina toiselle oliolle. lisäksi this-muuttujalla osoitetaan metodin sisällä nykyistä oliota.

```
function tulosta()
```

```
        {

  var tulos = "Hieno " + this.tyyppi + " " + this.malli
  + ", Omistaja " + this.omistaja);

        /* Tässä osoitetaan olioon Auto, jonka metodiksi funktio
         tulosta on määritelty */

        document.write( tulos,"<BR>");
       }



function Auto(tyyppi, malli, omistaja)
{
  this.tyyppi = tyyppi;
  this.malli = malli;
  this.omistaja = omistaja;
  this.tulosta = tulosta;
}

       /* Tässä osoitetaan funktioon Auto, kun muodostetaan
       olio-tyyppi Auto, jonka muuttujiksi määritellään tyyppi,
       malli ja omistaja sekä metodiksi tulosta. */
```

This-rakennetta voidaan hyödyntää myös lomakkeissa seuraavasti.

```
<INPUT TYPE="button" VALUE="ENTER" on-
Click="compute(this.form)">
```

Tässä rakenne **this.form** osoittaa lomakkeeseen, joka sisältää **button**-olion ja mahdollisesti muita oliota (text, textarea ym…). Lomakkeen muihin olioihin osoitetaan niiden nimellä tyyliin.

```
this.form.nimi
```

Yksittäisen olion muuttujan arvoon voidaan osoittaa seuraavasti.

```
this.form.nimi.value
```

3.10 Sisään rakennetut funktiot

JavaScriptiin on ohjelmoitu valmiiksi käteviä funktiota, joista ensimmäisenä esitellään funktiopari **escape()** ja **unescape()**. Näistä **escape()** palauttaa argumenttina annettavaa ASCII-merkkiä (ISO Latin-1) vastaavan luvun muodossa %xx, missä xx on vastaava luku. Seuraavan esimerkin palaute on “%26”.

```
escape("&");
```

Vastaavasti **unescape()**-funktio palauttaa argumenttina annettavaa lukua vastaavan ASCII-merkin. Luku on annettava muodossa “%xx”, missä xx on luku 0-255 kymmenkantaisena tai 0x0-0xFF heksalukuna. Seuraavan esimerkin palaute on “&”.

```
unescape("%26");
```

Seuraavaksi esitellään **eval()**-funktio. Sille annetaan argumentiksi matemaattinen lauseke, jonka arvon se palauttaa paluuarvonaan. Seuraavassa **tulos** niminen muuttuja saa arvokseen luvun 5.75.

```
tulos = eval (3*8/6 + 7/4);
```

Eval-funktiolle voidaan antaa myös merkkijono argumentiksi, jos se ei sisällä muita kuin matemaattisia merkkejä (poikkeuksena **Math**-olio). Se muuttaa merkit numeroiksi automaattisesti. Tämä ominaisuus on erittäin hyödyllinen luettaessa esimerkiksi form-rakenteen (joka sekin on olio) tekstikentästä tietoa, jota käsitellään aina merkkijonona. Argumentiksi voidaan antaa myös olio.

Esimerkki: Math-olio eval-funktion argumenttina

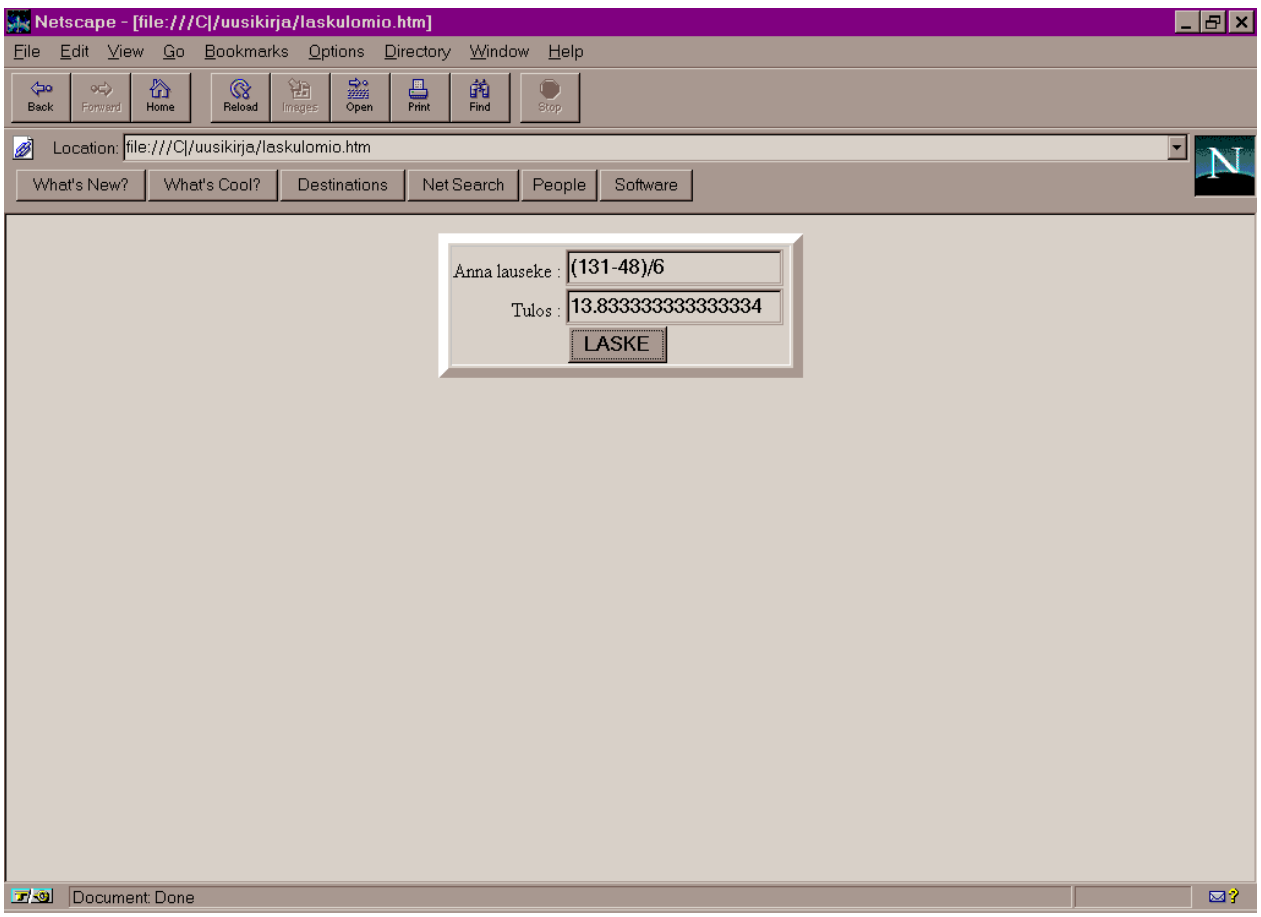
```
tulos = eval(3/4*Math.PI);
```

Tulos niminen muuttuja saa arvokseen 2.3561945. Math-olion PI-nimisestä muuttujasta löytyy pii:n arvo (3.1415927).

Seuraava Esimerkki lukee käyttäjän antaman merkkijonon (*jonka täytyy sisältää vain matemaattisia merkkejä kuten; kaikki numerot, +,/,*,(,-, jne. Tai Math-olion muuttujia ja metodeita*) syöttökentästä ja tulostaa vastauksen tulostuskenttään. Merkkijonon siirrossa tarvittaviin olioihin palataan myöhemmin.

```
<HTML>
<SCRIPT>

function laske(syote)    {

// laskun suoritus ja tulostus tulostukenttään

  syote.tulos.value = eval(syote.syotto.value)
}

</SCRIPT>
<BODY>
<CENTER>
<FORM NAME="laskulomake">
<TABLE border="8">
<TR align="right">
<TD>

Anna lauseke :
 <INPUT TYPE="text" NAME="syotto">
<BR>

Tulos : <INPUT TYPE="text" NAME="tulos">
<BR>
<CENTER>
```

```
<INPUT TYPE="button" VALUE="LASKE" on-
Click="laske(this.form)">

</CENTER></TD></TR></TABLE>
</CENTER>
</FORM>
</BODY></HTML>
```



Edellisen esimerkin tuloste.

Unix-järjestelmissä voidaan käyttää **isNan**-funktiota arviomaan sitä onko annettu argumentti luku. Se palauttaa joko arvon **true** (arvo=luku) tai **false** (arvo= ei luku). Syntaksi on seuraava.

```
isNan(testiarvo);
```

Lisäksi JavaScriptiin on rakennettu funktiopari **parseFloat()** ja **parseInt()** suorittamaan erilaisia lukumuunnoksia. **ParseFloat()**-funktio

palauttaa argumenttina annetusta merkkijonosta kaikki merkit siihen asti, kunnes vastaan tulee merkki, joka **ei ole** mikään seuraavista **+ - .Ee0123456789**. Jos ensimmäinen merkki ei ole mikään edellisistä, niin palautetaan 0. Seuraavan esimerkin kaikki variaatiot palauttavat arvon 3.14 .

```
parseFloat("3.14")
parseFloat("314e-2")
parseFloat("0.0314E+2)
var x = "3.14"
parseFloat(x)
```

Vastaavasti seuraava Esimerkki palauttaa arvon 0.

```
parseFloat("FF2")
```

Lukujärjestelmämuunnos saadaan aikaan **parseInt(**merkkijono, kanta-luku**)**-funktiolla. Jos annetussa merkkijonossa on merkki, joka ei ole numeerinen kyseisessä lukujärjestelmässä niin se ja kaikki sen jälkeiset merkit jätetään huomioimatta ja jos kyseessä on merkkijonon ensimmäinen merkki niin palautetaan 0. Funktio katkaisee muunnettavat luvut kokonaisluvuiksi. Kaikki seuraavan esimerkin variaatiot palauttavat luvun 15_{10}.

```
parseInt("F", 16)
parseInt("17", 8)
parseInt("15", 10)
parseInt(15.99, 10)
parseInt("FXX123", 16)
parseInt("1111", 2)
parseInt("15*3", 10)
```

Vastaavasti kaikki seuraavan esimerkin variaatiot palauttavat arvon 0.

```
parseInt("Hello", 8)
```

```
parseInt("0x7", 10)
parseInt("FFF", 10)
```

4. Ohjelmointi JavaScript-kielellä

Ohjelmointi perustuu JavaScriptistä löytyviin valmiisiin olioihin, (joilla on lukuisa joukko valmiita metodeita sekä muuttujia) sekä koodissa määriteltäviin omiin funktioihin. Näiden lisäksi voidaan määritellä myös uusia oliota.

JavaScript-koodia voidaan sisällyttää HTML-dokumenttiin kahdella tavalla: sijoittamalla koodi <SCRIPT> … </SCRIPT> tagien väliin tai soveltamalla niin sanottuja **tapahtumafunktioita**, jotka tunnistavat ja reagoivat tiettyihin tapahtumiin dokumentissa, esimerkiksi painonapin painamiseen. JavaScript-koodi voidaan ladata myös tiedostosta <SCRIPT>-tagia hyödyntäen seuraavasti. Tiedostoon kirjoitetaan siis pelkkä koodi, jollakin tarkoitukseen sopivalla editorilla (esim. Windowsin N*otepad*).

<SCRIPT [LANGUAGE=”JavaScript”] [SRC=
”tähän URL-osoite/omascript.js”] > …</SCRIPT>

Tallennetun koodin tiedoston tarkennin on *.js

<SCRIPT>-tagissa vaaditaan joko **LANGUAGE** tai **SRC** tunniste, jotta ohjelma tunnistaisi mitä koodia käytetään. Seuraavana on esimerkki JavaScript-koodin käytöstä HTML-dokumentissa.

```
<HTML>
<HEAD>
<!--  Piilota tämä osa vanhoilta browsereilta -- >
<SCRIPT LANGUAGE=”JavaScript”>
/*   Funktioiden määrittelyt kannattaa sijoittaa tähän */
</SCRIPT>
<!--  Lopeta piilottaminen  -- >
</HEAD>

<BODY>
<!--   Lomakkeet, taulukot, painonapit ym.  -- >
```

```
<! -- script-tagien sisällä olevan koodi kannatta
piilottaa vanhoilta
          netscape versioilta  -- >
<SCRIPT>
//  JavaScript-koodia
</SCRIPT>
<!--  Piilotus päättyy  -- >
<!--  mahdollinen HTML-koodi jatkuu  -- >
</BODY>
</HTML>
```

Tai

```
<HTML>
<HEAD>

<!--  Piilota tämä osa vanhoilta browsereilta -- >
<SCRIPT LANGUAGE=”JavaScript” SRC=”koodi.js”>
</SCRIPT>
<!--  Lopeta piilottaminen  -- >
</HEAD>

<BODY>
<!--   Lomakkeet, taulukot, painonapit ym.  -- >

<SCRIPT LANGUAGE=”JavaScript” SRC=”koodi_2.js >
</SCRIPT>

<!--  mahdollinen HTML-koodi jatkuu  -- >
</BODY>
</HTML>
```

4.1 Tapahtumafunktiot

Tapahtumafunktioiden *(Event Handlers)* koodi suoritetaan, kun kyseiseen tapahtumaan reagoiva olio on ohjelmoitu koodiin ja kyseinen tapahtuma tapahtuu. JavaScript-kielessä on 9 tapahtumafunktiota.

onBlur reagoi kursorin poistamiseen

	lomakkeen kentästä.
onClick	reagoi hiiren napin painallukseen.
onChange	reagoi lomakkeen kenttien sisällön muutokseen.
onFocus	reagoi kursorin asettamiseen lomakkeen kenttään.
onLoad	reagoi sivun alkulataukseen.
onMouseOver	reagoi hiiren kohdistimen viemiseen linkin päälle.
onSelect	reagoi lomakkeen kentän maalaamiseen.
onSubmit	reagoi lomakkeen lähettämiseen eteenpäin.
onUnload	reagoi sivulta poistumiseen.

Tapahtumafunktioita käytetään seuraavalla rakenteella.

```
<TAGI   tapahtumafunktio =”JavaScript-koodi”>
```

Esimerkki: Siirrettäessä kursori pois tekstikentästä, luodaan uusi ikkuna.

```
<INPUT type=”text” name=”naytto” on-
Blur=”window.open(‘http://kone/hakemisto/*.html’,’uusi’,)”>
```

Huomaa, että **window**-olion **open**-metodin argumentti-listassa ei voi käyttää (“ “)-merkkejä. Listassa pitää käyttää (‘ ‘)-merkkejä, koska yhtäsuuruus merkin jälkeen tuleva JavaScript-koodi pitää kirjoittaa lainausmerkkien (“ “) sisään.

4.2 Olioiden luonti

Valmiiden olioiden lisäksi voidaan JavaScript-kielessä määritellä mitä moninaisempia oliota funktioiden avulla. Määrittely aloitetaan kirjoittamalla funktio, joka määrittelee olion tyypin ja luomalla tätä tyyppiä vastaava olio komennolla **new**.

Esimerkki: luodaan olio-tyyppi **Auto**-funktiolla, jossa määritellään oliotyypille kolme muuttujaa, joiden nimiksi tulee **merkki**, **malli** ja **omistaja**. Muuttujien arvo määräytyy parametrien kautta, jotka annetaan oliota luodessa. Määrittelyssä käytetään hyväksi **this**-osoitinrakennetta.

```
function Auto(merkki, malli)
{
 this.merkki=merkki;
 this.malli=malli;
}
```

Nyt luodaan olio **myynti_ilmoitus** komennolla **new** seuraavasti.

```
myynti_ilmoitus = new Auto(“automerkki”, “vanha”) ;
```

Kun olion tyyppi on määritelty, voidaan saman tyyppisiä oliota luoda kuinka paljon tahansa yllä esitetyllä tavalla. Olion muuttujaan voidaan sijoittaa myös toinen olio seuraavalla tavalla. Määritetään ensin toinen olio-tyyppi nimeltään **henkilo.**

```
function henkilo(nimi, ika, sukupuoli)
{
  this.nimi = nimi ;
  this.ika = ika;
  this.sukupuoli = sukupuoli ;          }
```

Toiseksi luodaan kaksi uutta oliota nimiltään **Matti** ja **Pirkko**.

```
Matti = new henkilo(“Matti”, 30, “mies”);

Pirkko = new henkilo(“Pirkko”, 35, “nainen”);
```

Tämän jälkeen lisätään **Auto**-oliotyyppiin uuden muuttujan määrittely nimeltään **omistaja**. Lisäyksen jälkeen **Auto**-oliotyypin määrittely on tämän näköinen.

```
function Auto(merkki, malli, omistaja)
{
  this.merkki=merkki;
  this.malli=malli;
  this.omistaja=omistaja;
}
```

Nyt voidaan luoda **Auto**-oliotyyppisiä oliota, joiden muuttujiin voidaan sijoittaa toinen olio.

```
tama_auto = new Auto(“joku_merkki”, “uusi”, Matti);
tuo_auto = new Auto(“toinen_merkki”, “vähän käytetty”, Pirkko);
```

Olion **tuo_auto** muuttujan **nimi** sisältö saadaan selville tämän jälkeen seuraavasti.

```
var kuka = tuo_auto.omistaja.nimi;
```

```
document.write(“Omistaja : “, kuka);
```

Aikaisemmin luotuun olioon voi määritellä uuden muuttujan seuraavasti.

```
tama_auto.vari = “sininen”;
```

Lause määrittelee oliolle **tama_auto** uuden muuttujan yksinkertaisesti sijoittamalla siihen merkkijonon “sininen”. Tämä muuttuja on voimassa ainoastaan oliossa **tama_auto**. Jos halutaan kaikkiin saman tyyppisiin olioihin uusi muuttuja, niin se on lisättävä tyypin määrittelevään funktioon.

4.3 Metodin luominen

Metodi määritetään kullekin oliotyypille samaan tapaan kuin muuttujatkin, mutta ensin kirjoitetaan itse metodin toiminnan määrittelevä funktio. Seuraavassa esimerkissä funktiosta **nayta_kuva** tehdään **Auto**-oliotyypin metodi **nayta_kuva**.

```
function nayta_kuva()
{
  document.write("<IMG SRC=’http://kone/hakemisto/kuva.gif’
   ALIGN=left HSPACE=20 VSPACE=-20 >");

  var tulos = “Hieno “ + this.merkki + ” “ +  this.malli + “Omistaja : “ +
this.omistaja.nimi ;
  document.write(tulos);
}
```

Nyt lisätään metodin määrittely **Auto**-oliotyypin määrittelyfunktioon **Auto**, joka näyttää lisäyksen jälkeen tällaiselta.

```
function Auto(merkki, malli, omistaja)
{
  this.merkki=merkki;
  this.malli=malli;
  this.omistaja=omistaja;
  this.nayta_kuva = nayta_kuva ;
}
```

Tämän jälkeen uutta metodia voidaan kutsua koodissa esimerkiksi seuraavasti.

```
tuo_auto.nayta_kuva();
```

Esimerkki: tähän on koottu yllä oleva yhdeksi toimivaksi koodiksi.

```
<HTML>
<HEAD>
<!--  Piilota tämä  -- >
<SCRIPT LANGUAGE="JavaScript">

  function nayta_kuva(){

   // Metodin nayta_kuva määrittelyfunktio

   if(this.omistaja.nimi == “Pirkko”)

   {
      document.write("<IMG
SRC=http://osoite/kuva1.gif        ALIGN=left
HSPACE=20 VSPACE=-20 >");
   }


/* Huom !  Ohjelma saattaa antaa virheilmoituksia jos
document.write argumenttilista on kirjoitettu useam-
malle kuin yhdelle riville */
```

```
  if(this.omistaja.nimi == “Matti”)

   {
     document.write("<IMG SRC=http://osoite/kuva2.gif
ALIGN=left
     HSPACE=20 VSPACE=-20 >");

    }


   var tulos = "Hieno " + this.merkki + " " +
this.malli    +
   Omistaja : " +  this.omistaja.nimi ;

   document.write("<BIG>",tulos,"</BIG><BR><BR>");
   document.write("<BR><BR><BR><BR><HR><BR>");

   // yllä oleva rivi vaikuttaa vain sivun ulkoasuun
  }


  function Auto(merkki, malli, omistaja) {

   // Oliotyypin Auto määrittely
  this.merkki=merkki;
  this.malli=malli;
  this.omistaja=omistaja;
  this.nayta_kuva = nayta_kuva ;
 }


function henkilo(nimi,ika,sukupuoli){

  // Oliotyypin henkilo määrittely
  this.nimi = nimi ;
  this.ika = ika ;
  this.sukupuoli = sukupuoli ;
}

</SCRIPT>
<!-- Lopeta piilotus -- >
</HEAD>
<BODY>
<SCRIPT>
```

```
// Olioiden luonti
Matti = new henkilo("Matti", 30, "mies");
Pirkko = new henkilo("Pirkko", 35, "nainen");
tama_auto = new Auto(",auto_merkki", ",uusi,", Mat-
ti);
tuo_auto = new Auto(",eräs_auto_merkki", ",vähän käy-
tetty,", Pirkko);

tuo_auto.nayta_kuva() ;        // tulostus
tama_auto.nayta_kuva();

</SCRIPT>
</BODY>
</HTML>
```

4.4 Taulukkomuuttujan luominen

Taulukkomuuttujalla eli indeksoidulla muuttujalla osoitetaan nimettyyn muuttujaan, jota on monistettu tietty määrä ja sitten indeksoitu näin syntyneet muuttujat. Esimerkiksi.

```
apu[0]
apu[1]
apu[2]
```

Kaikkiin kolmeen apu-nimiseen indeksoituun muuttujaan voidaan sijoittaa eri arvo. Indeksoitu muuttuja luodaan olion avulla seuraavasti. Ensiksi luodaan **taulukko**-oliotyyppi.

```
function tee_taulu(n)
{
  this.pituus = n ;

  for(var i = 1 ; i < n ; i++)
   {  this[i] = 0 ;   }
  return this ;
}
```

Funktiolle **tee_taulu** syötetään parametrin **n** välityksellä haluttu muuttujamäärä. Ensimmäiseksi muuttujaksi tulee muuttuja **pituus**, jonka indeksi on täten [0] ja arvo = **n**. **For**-silmukalla luodaan uusia muuttujia (**n** -1) kappaletta, jotka nimetään samalla **this.1** , **this.2** , **this.3** jne. Näitä voidaan siis kutsua myös indeksoidusti **this[1]** , **this[2]** , **this[3]**. Jokaiseen muuttujaan sijoitetaan luku 0 alkuarvoksi.

Tämän jälkeen saadaan aikaan taulukko, jossa on kymmenen indeksoitua muuttujaa ja joiden alkuarvo on luku 0, luomalla **tee_taulu** tyyppinen olio seuraavasti. Huomaa, että ensimmäinen muuttuja (indeksillä [0]) on muuttuja **pituus**.

```
taulukko = new tee_taulu(11) ;
```

Nyt taulukon indeksoituihin muuttujiin voidaan sijoittaa haluttu arvo seuraavasti.

```
taulukko[1] = “minä” ;
taulukko[2] = “sinä” ;
taulukko[3] = “hän” ;
jne.
```

Muuttujassa **taulukko.1** on nyt merkkijono “minä” ja vastaavasti muuttujassa **taulukko. 2** merkkijono “sinä” jne. Samalla tavalla voidaan luoda myös indeksoituja olioita. Luodaan ensin oliotyyppi **kayttaja**.

```
function kayttaja(num, nimi, org)
{
  this.num = num ;
  this.nimi = nimi ;
  this.org = org ;
}
```

Tämän jälkeen luodaan indeksoituja olioita seuraavasti.

```
taulukko = new tee_taulu(3) ;
taulukko[1] = new kayttaja(1, “minä”, “firma Oy”) ;
taulukko[2] = new kayttaja(2, “sinä”, “company ltd”) ;
```

Ensiksi luodaan **tee_taulu** tyyppinen olio nimeltään **taulukko**, jonka jälkeen sijoitetaan **taulukko**-olion toiseen muuttujaan (huom. Indeksi [1]) nimeltään **1**. Uusi **kayttaja** tyyppinen olio (jonka nimeksi tulee myös **1**), jonka muuttujiin (**num**, **nimi** ja **org**) sijoitetaan arvot 1, “minä” ja “firma Oy”. Sama toistetaan **taulukko**-olion muuttujan **2** suhteen.
Näin menetellen saadaan suuretkin muuttujamäärät tulostettua lyhyellä koodilla **for**-silmukan avulla.

```
var n = 3 ;  // tässä tapauksessa 1+2 muuttujaa

for(var i=1 ; i<n ; i++)
{
  document.write(taulukko[i].num, “ ,“);
  document.write(taulukko[i].nimi, “ ,“);
  document.write(taulukko[i].org, “<BR><BR> “);
}
```

Tämä tuottaa tulosteen:

```
1 ,minä ,firma Oy
2 ,sinä ,company ltd
```

4.5 Math-olio

Sisään rakennetulla **Math**-olion metodeilla ja muuttujilla ratkaistaan matemaattisia ongelmia. Esimerkiksi **Math**-olion **PI**-muuttujaan on sijoitettu pii:n arvo. Jota voidaan käyttää esimerkiksi seuraavasti.

```
var ala = eval(Math.PI * r * r);
```

Yleisesti rakenne on seuraava.

```
Math.muuttuja_nimi
```

tai

```
Math.metodi_nimi(parametrit)
```

Muuttujat:

Math.E	e eli Neperin luku = 2.71828…
Math.LN10	luvun 10 luonnollinen logaritmi
Math.LN2	luvun 2 luonnollinen logaritmi
Math.SQRT1_2	neliöjuuri ½
Math.SQRT2	neliöjuuri 2

Metodit:

Math.abs(argumentti)	itseisarvo
Math.acos(argumentti)	arcus cosini, kulma (rad)
Math.asin(argumentti)	arcus sini kulma, (rad)
Math.atan(argumentti)	arcus tangentti, kulma (rad)
Math.ceil(argumentti)	pyöristys lähimpään suurempaan kokonaislukuun

Math.cos(argumentti)	cosini (rad)
Math.exp(argumentti)	e potenssiin argumentti
Math.floor(argumentti)	pyöristys lähimpään pienempään kokonaislukuun
Math.log(argumentti)	luonnollinen logaritmi argumentista
Math.max(arg1, arg2)	palauttaa suuremman kahdesta argumentista
Math.min(arg1, arg2)	palauttaa pienemmän kahdesta argumentista
Math.pow(arg1, arg2)	argumentti1 potenssiin argumentti2
Math.random	palauttaa satunnaisluvun väliltä 0-1. Toimii toistaiseksi vain X-window (UNIX) ympäristössä.

Seuraavalla funktiolla saadaan aikaan satunnaisluku väliltä 0-1 ja sillä voidaan korvata siten **random()**-metodi.

```
function SatLuku(){

  paivays = new Date();
  satluku= Math.abs(Math.sin(paivays.getTime()));
  return satluku;
}
```

Math.round(argumentti)	pyöritys x.5:sta ylöspäin ja x.4:sta alaspäin

Math.sin(argumentti)	sini (rad)
Math.sqrt(argumentti)	argumentin neliöjuuri
Math.tan(argumentti)	tangentti (rad)

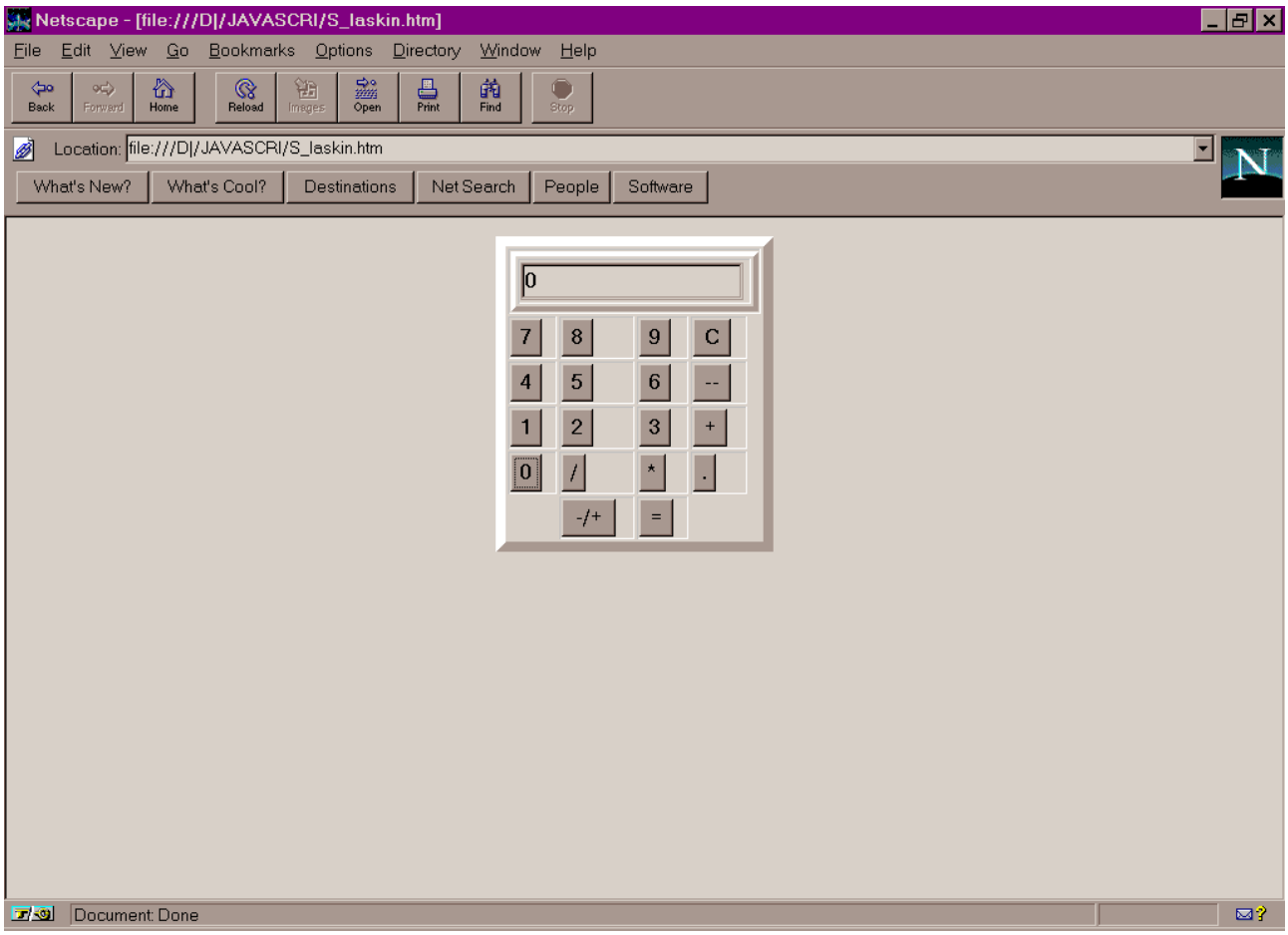
Esimerkki. Yksinkertainen Laskin.

```
<HTML>

<HEAD>
<!-- Piilota -->
<SCRIPT LANGUAGE="JavaScript">

var valmis = false
var puolet = false

function naytolle(laskin, arvo)
 {

       laskin.naytto.value = arvo;


      }

function valitys(syote, merkki)
    {
        if(merkki != "=" && merkki != "C")
         syote.value += merkki ;


          if(merkki == "=")
        {    arvo = eval(syote.value) ;
             naytolle(syote.form, arvo) ;  }
                }


function pyyhi(syote)
   {
```

```
       syote.value = syote.value.substring(0, syo-
te.value.length - 1) ;

           }



function muutamerkki(syote)
        {
            syote.value = 0 - syote.value
              }

<!--  -->
</SCRIPT>

</HEAD>

<BODY>
<CENTER>

<form>

<table border="8">
<tr align="center">
<td colspan="6">
<table border="4">
<tr>
<td>
<input name="naytto" value="0" defaultValue="0">
</td></tr></table>

<tr>
<td>
<input type="button" value=" 7 "  on-
Click="valitys(this.form.naytto, '7')">
</td>
<td>
<input type="button" value=" 8 "  on-
Click="valitys(this.form.naytto, '8')">
</td>
<td>
<input type="button" value=" 9 "  on-
Click="valitys(this.form.naytto, '9')">
</td>
<td>
<input type="button" value=" C "  on-
Click="pyyhi(this.form.naytto)">
```

```
</td></tr>
<tr><td>
<input type="button" value=" 4 "  on-
Click="valitys(this.form.naytto, '4')">
</td>
<td>
<input type="button" value=" 5 "  on-
Click="valitys(this.form.naytto, '5')">
</td>
<td>
<input type="button" value=" 6 "  on-
Click="valitys(this.form.naytto, '6')">
</td>
<td>
<input type="button" value=" -- "  on-
Click="valitys(this.form.naytto, '-')">
</td></tr>
<tr>
<td>
<input type="button" value=" 1 "  on-
Click="valitys(this.form.naytto, '1')">
</td>
<td>
<input type="button" value=" 2 "  on-
Click="valitys(this.form.naytto, '2')">
</td>

<td>
<input type="button" value=" 3 "  on-
Click="valitys(this.form.naytto, '3')">
</td>
<td>
<input type="button" value=" + "  on-
Click="valitys(this.form.naytto, '+')">
</td></tr>
<tr><td>
<input type="button" value=" 0 "  on-
Click="valitys(this.form.naytto, '0')">
</td>
<td>
<input type="button" value=" / "  on-
Click="valitys(this.form.naytto, '/')">
</td>
<td>
<input type="button" value=" * "  on-
Click="valitys(this.form.naytto, '*')">
</td>
```

```
<td>
<input type="button" value=" . "  on-
Click="valitys(this.form.naytto, '.')">
</td></tr>
<tr>
<td> </td>
<td>
<input type="button" value=" -/+ "  on-
Click="muutamerkki(this.form.naytto)">
</td>
<td>
<input type="button" value=" = "  on-
Click="valitys(this.form.naytto, '=')">
</td>
<td> </td>
</tr>
</td></tr></table>

</form>
</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

Esimerkki. Monitoiminen laskin.

```
<HTML>
<HEAD>
<!-- Hide from old browsers -->
<SCRIPT LANGUAGE="JavaScript">


function valitys(syote, merkki)
    {

    if(merkki != "=" && merkki != "C" && merkki != "x" &&
merkki != "sqxx" && merkki != "e" && merkki != "ex" &&
merkki != "LN" && merkki != "LOG")
        syote.value += merkki ;
```

```
    if(merkki == "pii")
        syote.value = Math.PI ;


    if(merkki == "DEG")
        syote.value = "DEG";

    if(merkki == "RAD")
         syote.value = "RAD";

    if(merkki == "=")     // lasku
        {

         var ptesti = false;
         var p = "p" ;

  for (var i = 0 ; i < syote.value.length ; i++) // syot-
  teen hajoitus merkeiksi
             {
         var  a = i;
         var b = 1
         b += i;


    if (p == syote.value.substring(a,b)) // tarkistus on-
ko x pot y
       { var apupot2 = syote.value.substring(0,a) ;
         var apupot3 = syote.value.substring(a +
1,syote.value.length);
         syote.value = Math.pow(apupot2,apupot3);
         i = syote.value.length ;
         ptesti = true }
                            }

    if(!ptesti)  // jos ei x pot y toimitaan normaalisti
         syote.value = eval(syote.value);

                                  }

    if(merkki == "x")
        syote.value = Math.pow(syote.value, 2);

    if(merkki == "sqxx")
        syote.value = Math.sqrt(syote.value);

    if(merkki == "e")
```

```
        syote.value = Math.E ;

    if(merkki == "ex")
        syote.value = Math.pow(Math.E, syote.value);

    if(merkki == "LN")
        syote.value = Math.log(syote.value);

    if(merkki == "LOG")
        syote.value = Math.log(syote.value)/Math.LN10 ;



            }


function trigovalitys(syote, tila, merkki)
    {
    if(tila.value == "DEG") {

    if(merkki == "sini") {

    var apurad = eval(syote.value/(180/Math.PI));
    syote.value = Math.sin(apurad);
                        }

    if(merkki == "cosini") {


    var apurad2 = eval(syote.value/(180/Math.PI));
    syote.value = Math.cos(apurad2);
                        }


    if(merkki == "tang") {

    var apurad3 = eval(syote.value/(180/Math.PI));
    if(apurad3 == (Math.PI/2) || apurad3 ==
((3/2)*(Math.PI)))
// likiarvo virheen korjaus
    {  syote.value = "Error";}
    else
    syote.value = Math.tan(apurad3);
                        }

    if(merkki == "asini") {
```

```
    var apurad4 = Math.asin(syote.value);
    syote.value = eval(apurad4*(180/Math.PI));
                      }


    if(merkki == "acosini") {

    var apurad5 = Math.acos(syote.value);
    syote.value = eval(apurad5*(180/Math.PI));
                      }


    if(merkki == "atang") {

    var apurad6 = Math.atan(syote.value);
    syote.value = eval(apurad6*(180/Math.PI));
                         }
                            }


    if(tila.value == "RAD") {

    if(merkki == "sini")
        {
      syote.value = Math.sin(syote.value);
                     }

    if(merkki == "cosini") {

    if(syote.value == (Math.PI/2) || syote.value ==
((3/2)*(Math.PI)))
 // likiarvo virheen korjaus
      syote.value = "0";
    else
      syote.value = Math.cos(syote.value);
                   }

    if(merkki == "tang") {

    if(syote.value == (Math.PI/2) || syote.value ==
((3/2)*(Math.PI)))
 // likiarvo virheen korjaus
    {  syote.value = "Error";}
    else
      syote.value = Math.tan(syote.value);}
```

```
    if(merkki == "asini") {

      syote.value = Math.asin(syote.value);
                }

    if(merkki == "acosini") {

      syote.value = Math.acos(syote.value);
                        }

    if(merkki == "atang") {

      syote.value = Math.atan(syote.value);
                }
                     }
                          }

function pyyhi(syote,tunnus) {

    if(tunnus == "C")
     syote.value = syote.value.substring(0, syo-
te.value.length - 1) ;

    if(tunnus == "CE")
    syote.value = "0";

                  }



function muutamerkki(syote)
        {
            syote.value = 0 - syote.value
              }


function xpoty(syote)
        {
           syote.value += "p";
                }



<!-- stop hiding -->
</SCRIPT>
```

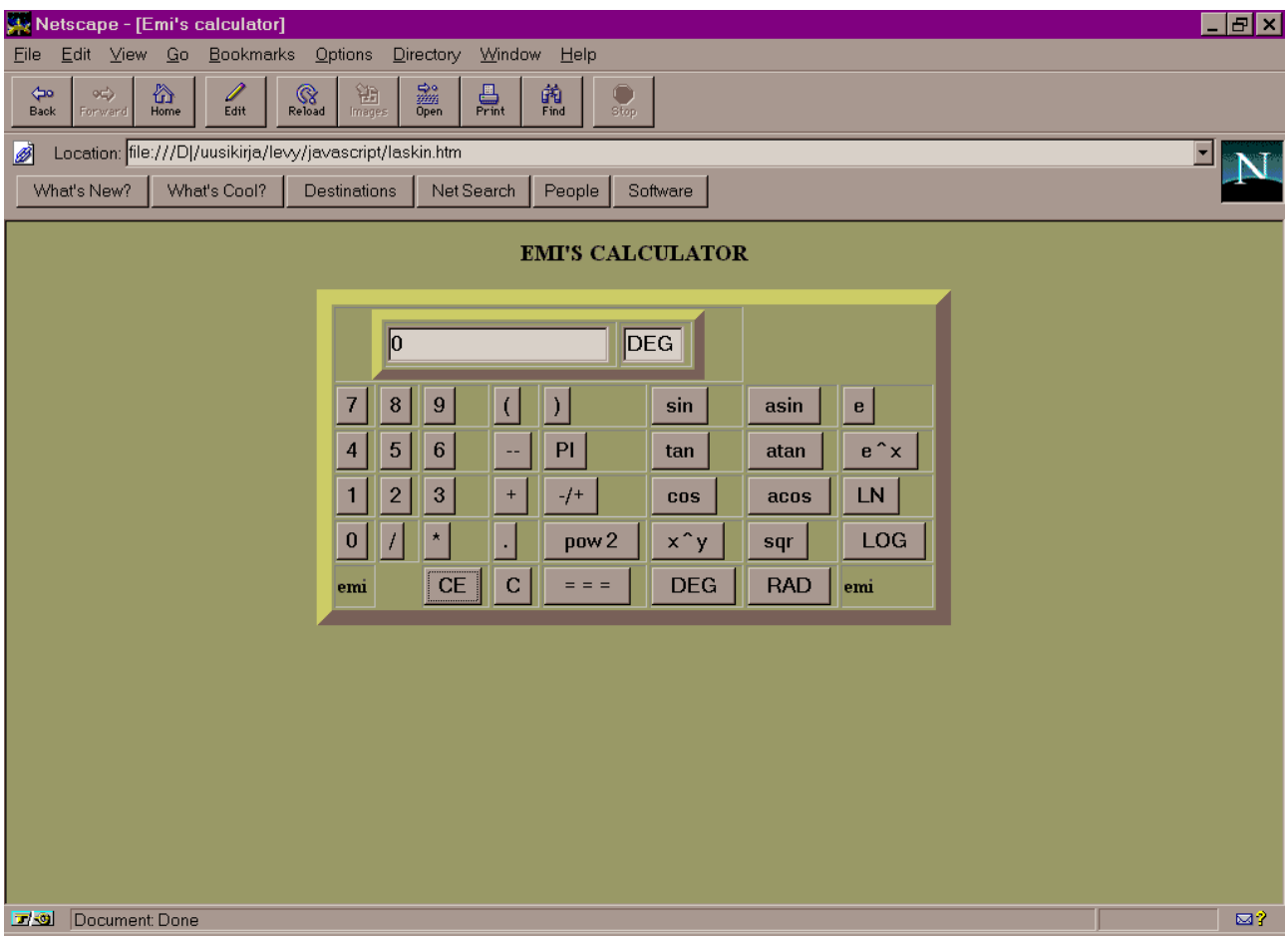
```
</HEAD>
<BODY bgcolor="#9,8,6">
<TITLE>Emi's calculator</TITLE>
<CENTER><BIG><B>EMI'S CALCULATOR</B></BIG></CENTER>

<CENTER>
<form>

<table border="12">
<tr align="center">
<td colspan="6">
<table border="8">
<tr>
<td>
<input type="text" name="naytto" value="0" >
</td>
<td>
<input type="text" size=4 name="tila" value="DEG" >
</td></tr></table>

<tr>
<td>
<input type="button" value=" 7 "  on-
Click="valitys(this.form.naytto, '7')">
</td>
<td>
<input type="button" value=" 8 "  on-
Click="valitys(this.form.naytto, '8')">
</td>
<td>
<input type="button" value=" 9 "  on-
Click="valitys(this.form.naytto, '9')">
</td>
<td>
<input type="button" value=" ( "  on-
Click="valitys(this.form.naytto, '(')">
</td>
<td>
<input type="button" value=" ) "  on-
Click="valitys(this.form.naytto, ')')">
</td>
<td>
<input type="button" value=" sin "  on-
Click="trigovalitys(this.form.naytto, this.form.tila,
'sini')">
</td>
<td>
```

```
<input type="button" value=" asin "  on-
Click="trigovalitys(this.form.naytto, this.form.tila,
'asini')">
</td>
<td>
<input type="button" value=" e "  on-
Click="valitys(this.form.naytto, 'e')">
</td>
</tr>
<tr><td>
<input type="button" value=" 4 "  on-
Click="valitys(this.form.naytto, '4')">
</td>
<td>
<input type="button" value=" 5 "  on-
Click="valitys(this.form.naytto, '5')">
</td>
<td>
<input type="button" value=" 6 "  on-
Click="valitys(this.form.naytto, '6')">
</td>
<td>
<input type="button" value=" -- "  on-
Click="valitys(this.form.naytto, '-')">
</td>
<td>
<input type="button" value=" PI "  on-
Click="valitys(this.form.naytto, 'pii')">
</td>
<td>
<input type="button" value=" tan "  on-
Click="trigovalitys(this.form.naytto, this.form.tila,
'tang')">
</td>
<td>
<input type="button" value=" atan "  on-
Click="trigovalitys(this.form.naytto, this.form.tila,
'atang')">
</td>
<td>
<input type="button" value=" e ^ x "  on-
Click="valitys(this.form.naytto, 'ex')">
</td>
</tr>
<tr>
<td>
```

```
<input type="button" value=" 1 "  on-
Click="valitys(this.form.naytto, '1')">
</td>
<td>
<input type="button" value=" 2 "  on-
Click="valitys(this.form.naytto, '2')">
</td>

<td>
<input type="button" value=" 3 "  on-
Click="valitys(this.form.naytto, '3')">
</td>
<td>
<input type="button" value=" + "  on-
Click="valitys(this.form.naytto, '+')">
</td>
<td>
<input type="button" value=" -/+ "  on-
Click="muutamerkki(this.form.naytto)">
</td>
<td>
<input type="button" value=" cos "  on-
Click="trigovalitys(this.form.naytto, this.form.tila,
'cosini')">
</td>
<td>
<input type="button" value=" acos "  on-
Click="trigovalitys(this.form.naytto, this.form.tila,
'acosini')">
</td>
<td>
<input type="button" value=" LN "  on-
Click="valitys(this.form.naytto, 'LN')">
</td>
</tr>
<tr><td>
<input type="button" value=" 0 "  on-
Click="valitys(this.form.naytto, '0')">
</td>
<td>
<input type="button" value=" / "  on-
Click="valitys(this.form.naytto, '/')">
</td>
<td>
<input type="button" value=" * "  on-
Click="valitys(this.form.naytto, '*')">
</td>
```

```
<td>
<input type="button" value=" . "  on-
Click="valitys(this.form.naytto, '.')">
</td>
<td>
<input type="button" value=" pow 2 "  on-
Click="valitys(this.form.naytto, 'x')">
</td>
<td>
<input type="button" value=" x ^ y "  on-
Click="xpoty(this.form.naytto, 'xy')">
</td>
<td>
<input type="button" value=" sqr "  on-
Click="valitys(this.form.naytto, 'sqxx')">
</td>
<td>
<input type="button" value=" LOG "  on-
Click="valitys(this.form.naytto, 'LOG')">
</td>
</tr>
<tr>
<td>
<B>emi</B>
</td>
<td>
</td>
<td>
<input type="button" value=" CE " on-
Click="pyyhi(this.form.naytto,'CE')">
</td>
<td>
<input type="button" value=" C "  on-
Click="pyyhi(this.form.naytto,'C')">
</td>
<td>
<input type="button" value=" = = = "  on-
Click="valitys(this.form.naytto, '=')">
</td>
<td>
<input type="button" value=" DEG "  on-
Click="valitys(this.form.tila, 'DEG')">
</td>
<td>
<input type="button" value=" RAD "  on-
Click="valitys(this.form.tila, 'RAD')">
</td>
```

```
<td>
<B>emi</B>
</td>
</tr>
</td></tr></table>

</form></CENTER>

</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

4.6 String-olio

Aina kun muuttujaan sijoitetaan merkkijono, tulee muuttujasta automaattisesti **String**-tyyppinen olio.

```
var kuka = “minä” ;
```

Yllä olevalla sijoituslauseella saadaan aikaan **String**-tyyppinen olio nimeltään **kuka**. Sitä voidaan käsitellä kuten muuttujia yleensä, ottaen huomioon, että sisältönä on merkkijono. Myös pelkkä merkkijono lauseke on **String**-tyyppinen olio. Kuten.

```
“Hyvää päivää WWW”
```

String-oliotyypillä on lukuisa joukko käyttökelpoisia metodeita ja yksi muuttuja. Yleinen rakenne on:

```
merkkijono.muuttuja

tai

merkkijono.metodi()
```

Muuttujat:

length	muuttujaan on sijoitettu merkkijonon pituus lukuna.

```
var lukumaara = “Hyvää päivää”.length ;
```

Muuttujaan **lukumaara** sijoitetaan luku 12. Myös tyhjä merkki lasketaan mukaan.

```
var lukumaara = kuka.length ;
```

Nyt muuttujaan **lukumaara** sijoitetaan luku 4, kun muuttujaan **kuka** oli sijoitettu merkkijono “minä”. Metodit ovat:

merkkijono.anchor(arg)	Argumentista tehdään ankkuri <aname=”arg“>merkkijono ja merkkijonosta otsikko. Katso **Esimerkki** metodin **link** kohdalta.
merkkijono.big(arg)	Palauttaa argumenttina olevan merkkijonon isolla fontilla.

Esimerkiksi seuraavalla lauseella tulostetaan muuttujan **teksti** sisältö isolla fontilla.

```
var teksti = “Paljon tekstiä “ ;
document.write(teksti.big(teksti));
```

Vastaa HTML-tageja <BIG> … </BIG> .

merkkijono.blink(arg)	Palauttaa argumenttina olevan merkkijonon vilkkuvana.

Esimerkiksi seuraavalla lauseella tulostetaan edellisen kohdan muuttujan **teksti** sisältö isolla fontilla ja vilkkuvana.

```
document.write(teksti.big(teksti).blink(teksti));
```

merkkijono.bold(arg)	Palauttaa argumenttina olevan merkkijonon lihavoituna. Käytetään kuten metodeita **blink** ja **bold**.
merkkijono.charAt(indeksi)	Palauttaa indeksiä vastaavan merkin merkkijonosta. Huomaa, että ensimmäisen merkin indeksi on 0.

Esimerkiksi merkki *s* tulostetaan seuraavasti.

```
var teksti = “Tekstiä “ ;
document.write(teksti.charAt(3));
```

merkkijono.fixed() Palauttaa merkkijonon fixed-fontilla. Vastaa HTML-tageja <fixed> … </fixed>.

merkkijono.fontcolor(#0x)
Määrää merkkijonon fontin värin. Väri ilmoitetaan heksa- tai desimaalilukuna.**Esimerkki**. Desimaalilukusyntaksi (”#9,8,6”). Heksalukusyntaksi (”#00FF00”). Värit voi myös kirjoittaa suoraan (*englanniksi*), esimerkiksi **fontcolor(yellow)**.

Esimerkiksi merkkijonon “teksti” halutaan tulostuvan punaisena.

```
document.write(“teksti”.fontcolor(“#ff0000”));
```

Seuraavassa muutamien perusvärien koodeja heksalukuina:

“#000000”	musta
“#0000FF”	sininen
“#00FF00”	vihreä
“#FF0000”	punainen
“#FFFF00”	keltainen
“#00FFFF”	vaalean sininen
“#AAAAAA”	hopean harmaa

“#FFFFFF”	valkoinen

merkkijono.fontsize(arg) Palauttaa merkkijonon määrätyn kokoisella fontilla.Vastaa HTML-tageja <FONT size = ”arg”> … </FONT>. Argumentti väliltä 1-7.

merkkijono.indexOf(arg1,[arg2])

Palauttaa merkkijonon määrätyn merkin indeksin. Argumentiksi 1 annetaan merkki, jonka indeksi halutaan selvittää. Argumentiksi annetaan 2 indeksi josta etsintä halutaan aloittaa (jos jätetään pois, niin aloitetaan alusta).

Esimerkiksi halutaan tulostaa merkkijonon **teksti** merkin **k** indeksi ja etsintä aloitetaan ensimmäisestä merkistä. Jos etsittävää merkkiä ei löydy palautetaan **-1** .

```
document.write(“teksti”.indexOf(“k”));
```

Huomaa, että merkkijonosta palautetaan ensimmäisen merkin indeksi, joka vastaa etsittävää. Jos halutaan tulostaa merkkijonon **teksti** jälkimmäisen **t** ;n indeksi, niin käytetään esimerkiksi seuraavaa lausetta. Vertaa **lastIndexOf**-metodiin.

```
document.write(“teksti”.indexOf(“t”,1));
```

merkkijono.italics() Palauttaa merkkijonon *italics*-fontilla.

merkkijono.lastIndexOf(arg1,[arg2])
Kuten metodi **indexOf**, mutta etsintä aloitetaan merkkijonon lopusta.

Esimerkiksi tulostetaan merkkijonon **teksti** jälkimmäisen **t** ;n indeksi.

```
document.write(“teksti”.lastIndexOf(“t”));
```

merkkijono.link(arg) Argumentista (URL-osoite) tehdään linkki ja merkkijonosta linkin teksti.

Esimerkki 1:

```
var linkki = “mene “ ;
document.write(linkki.link(“http://osoite” );

tai

document.write(“mene”.link(“http://osoite” );
```

Tuottaa saman lopputuloksen eli näytölle tulostuu hypertekstilinkki, jonka tekstiksi tulee **mene** ja osoitteeksi **http://osoite**.

Esimerkki 2:

```
<SCRIPT>
document.write(“mene”.link(“#tuonne”));
// JavaScript-koodia
</SCRIPT>
<!--  HTML-koodia  -- >
<SCRIPT>
```

```
document.write(“Otsikko”.anchor(“tuonne”));
// JavaScript-koodia
</SCRIPT>
```

Näyttöön tulostuu hypertekstilinkki, jonka tekstiksi tulee **mene** ja osoitteeksi ankkuri **tuonne** ja siirtymiskohtaan tulostuu teksti **Otsikko**. Tämä vastaa seuraavaa HTML-koodia.

```
<A HREF=”#tuonne”>mene</A>
<--  HTML-koodia  - ->
<A NAME=”tuonne”>Otsikko</A>
```

merkkijono.small() Palauttaa merkkijonon pienellä fontilla. Vastaa HTML-tageja <SMALL> …</SMALL>.

merkkijono.strike() Palauttaa merkkijonon fontilla, jossa ~~tekstin~~ läpi kulkee ~~viiva~~.

merkkijono.sub() Palauttaa merkkijonon alaindeksinä. Katso metodi **sup**.

merkkijono.substring(a,b)

Palauttaa merkkijonosta osan alkaen **a**;sta ja päättyen ennen **b**;tä , jos **a** on pienempi kuin **b**. Jos **a** on suurempi kuin **b**, niin palauttaa osan alkaen **b**;stä ja päättyen ennen **a**:ta. **A**;lla ja **b**;llä osoitetaan yksittäisen merkin indeksiä (ensimmäinen = 0).

Esimerkiksi halutaan tulostaa merkkijonosta **Tämä on koe** osa **koe** eli indeksit 8,9 ja 10. Huomaa, että myös tyhjät merkit lasketaan.

```
var teksti = “Tämä on koe” ;
document.write(teksti.substring(8,11));
```

tai

```
document.write(teksti.substring(11,8));
```

merkkijono.sup() Palauttaa merkkijonon yläindeksinä.

Esimerkki:

```
document.write("x","2".sup());
document.write("<BR>x","2".sub());
```

Tuottaa tulosteen:

x^2
x_2

merkkijono.toLowerCase()
Palauttaa isoilla kirjaimilla kirjoitetun merkkijonon pienillä kirjoitettuna.

merkkijono.toUpperCase()
Palauttaa pienillä kirjaimilla kirjoitetun merkkijonon isoilla kirjoitettuna.

Esimerkki. Dokumentti, jossa on kaksi kehystä (*Frames*). Toisessa valitaan teksti, sen tyyli, koko ja väri. Toiseen tulostetaan valinnat.

1. Kehysten määrityssivu (string.htm).

```
<HTML>
<HEAD>
<TITLE>String-test</TITLE>
</HEAD>

<FRAMESET COLS="50%,*">
        <FRAME SRC="ohjaus.htm" NAME="ohjaus">
        <FRAME SRC="tulos.htm" NAME="tulos">
</FRAMESET>
</HTML>
```

2. Aluksi tulostus kehykseen tulostettava sivu (tulos.htm).

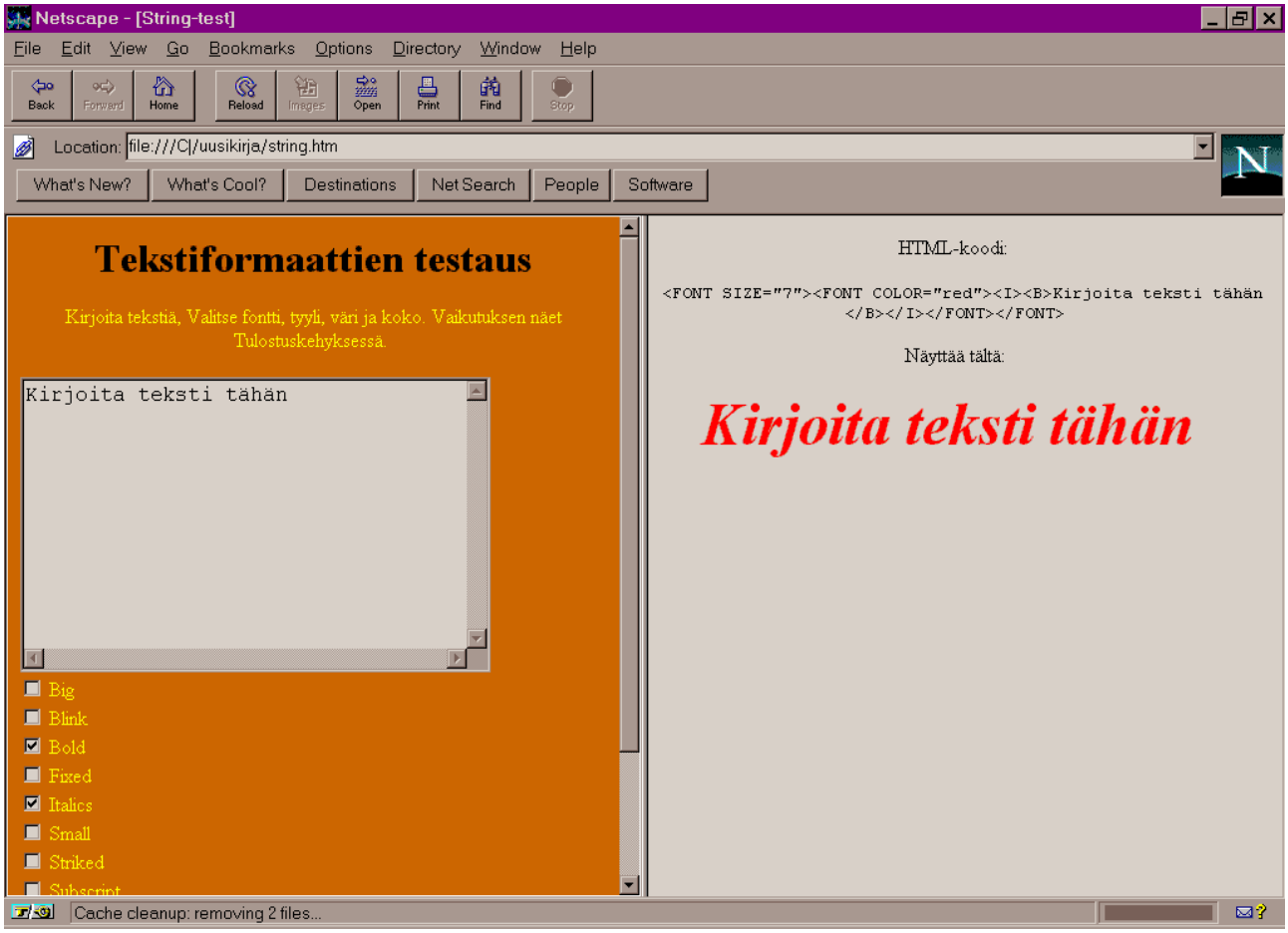
```
<HTML>
<BODY>

</BODY>
</HTML>
```

3. Varsinainen toiminta sivu (ohjaus.htm).

```
<HTML>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">
<!-- Piilotus -->

  function nayta(form) {
     var tyyli = form.naytto.value;
```

```
     tyyli = (form.big.checked) ? tyyli.big() : tyyli;
     tyyli = (form.blink.checked) ? tyyli.blink() : tyy-
li;
     tyyli = (form.bold.checked) ? tyyli.bold() : tyyli;
     tyyli = (form.fixed.checked) ? tyyli.fixed() : tyy-
li;
     tyyli = (form.italics.checked) ? tyyli.italics() :
tyyli;
     tyyli = (form.small.checked) ? tyyli.small() : tyy-
li;
     tyyli = (form.strike.checked) ? tyyli.strike() :
tyyli;
     tyyli = (form.sup.checked) ? tyyli.sup() : tyyli;
     tyyli = (form.sub.checked) ? tyyli.sub() : tyyli;
     tyyli = (form.vari.value == "") ? tyy-
li.fontcolor("black") : tyyli.fontcolor(form.vari.value);
     tyyli = (form.koko.value == "") ? tyyli.fontsize(3)
: tyyli.fontsize(form.koko.value);

     var L_tulos = "<CENTER>HTML-koodi: <XMP>";
     L_tulos += tyyli;
     L_tulos += "</XMP> Näyttää tältä:<P>"

//XMP-tagi näyttää HTML-koodin sellaisenaan

     L_tulos += tyyli;
     L_tulos += "</CENTER>";

     parent.tulos.document.open("text/html");
     parent.tulos.document.write(L_tulos);
     parent.tulos.document.close();

   }

   // Lopeta piilotus -->
</SCRIPT>

<BODY BGCOLOR="c95c14">

<CENTER>
<H1>Tekstiformaattien testaus</H1><font color="#ffff00">
Kirjoita tekstiä, Valitse fontti, tyyli, väri ja koko.
Vaikutuksen näet
Tulostuskehyksessä.
<BR>
</CENTER>
```

```
<FORM>

<TEXTAREA NAME="naytto" ROWS=10 COLS=35 onChan-
ge="nayta(this.form);">
Kirjoita teksti tähän
</TEXTAREA><BR>

   <INPUT TYPE="checkbox" NAME="big" on-
Click="nayta(this.form);">Big<BR>
   <INPUT TYPE="checkbox" NAME="blink" on-
Click="nayta(this.form);">Blink<BR>
   <INPUT TYPE="checkbox" NAME="bold" on-
Click="nayta(this.form);">Bold<BR>
   <INPUT TYPE="checkbox" NAME="fixed" on-
Click="nayta(this.form);">Fixed<BR>
   <INPUT TYPE="checkbox" NAME="italics" on-
Click="nayta(this.form);">Italics<BR>
   <INPUT TYPE="checkbox" NAME="small" on-
Click="nayta(this.form);">Small<BR>
   <INPUT TYPE="checkbox" NAME="strike" on-
Click="nayta(this.form);">Striked<BR>
   <INPUT TYPE="checkbox" NAME="sub" on-
Click="nayta(this.form);">Subscript<BR>
   <INPUT TYPE="checkbox" NAME="sup" on-
Click="nayta(this.form);">SuperScript<BR>

Fontin Väri: <INPUT TYPE="text" NAME="vari" VALUE="black"
onChange="nayta(this.form);"><BR>
Fontin koko (1 - 7): <INPUT TYPE="text" NAME="koko"
VALUE="3" onChange="nayta(this.form);">

</FORM>
<SCRIPT LANGUAGE="JavaScript">

   nayta(document.forms[0]);

</SCRIPT>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

4.7 Date-olio

Date-oliotyypillä käsitellään aikaa ja päivämääriä. Sillä on useita metodeita, mutta ei yhtään muuttujaa. Aikaa käsitellään JavaScriptissä hyvin samankaltaisesti kuin Javassa. Aika mitataan **millisekunteina** ajanhetkestä **1.1.1970 klo 00:00:00**. Päivämääriä, jotka ajoittuvat ennen tätä ajankohtaa ei voida toistaiseksi käsitellä. **Date**-tyyppinen olio luodaan seuraavasti.

```
uusiolio = new Date(parametrit) ;
```

Uusiolio voi olla myös olemassa olevan olion muuttuja. Parametreillä on seuraava vaikutus uuteen olioon.

- **Ei mitään** Luo nykyisen päivämäärän ja kellon ajan

Esimerkki: (20.1.1996 klo 02:21:30 La)

```
tanaan = new Date();
document.write(tanaan);
```

Tuottaa tulosteen:

```
Sat Jan 20 02:21:30 1996
```

- **merkkijono** Päiväys esitetään seuraavassa muodossa.

Kuukausi (*English*) **päivä, vuosi tunnit:minuutit:sekunnit** .

Esimerkki:

```
vappu_96 = new Date("May 1, 1996 12:00:00");
document.write(vappu_96);
```

Tuottaa tulosteen:

```
Wed May 01 13:00:00 1996
```

Jos tunnit, minuutit ja sekunnit jätetään pois tulostuu seuraava merkkijono.

Wed May 01 01:00:00 1996

- **lukusarja** **E**dustavat vuotta, kuukautta ja päivää.

Esimerkki:

```
vappu_96 = new Date(96,4,1);
document.write(vappu_96);
```

Tuottaa saman tulosteen kuin edellinen esimerkki. Huomaa numerointi.

Sekunnit ja minuutit 0 - 59.
Tunnit 0 - 23.
Viikon päivä 0 - 6.
Kuukauden päivä 1- 31.
Kuukaudet 0 - 11.
Vuodet vuodesta 1900 (kahdella numerolla).

Lukusarjaan voidaan lisätä myös tunnit, minuutit ja sekunnit.

```
vappu_96 = new Date(96,4,1,20,30,0);
```

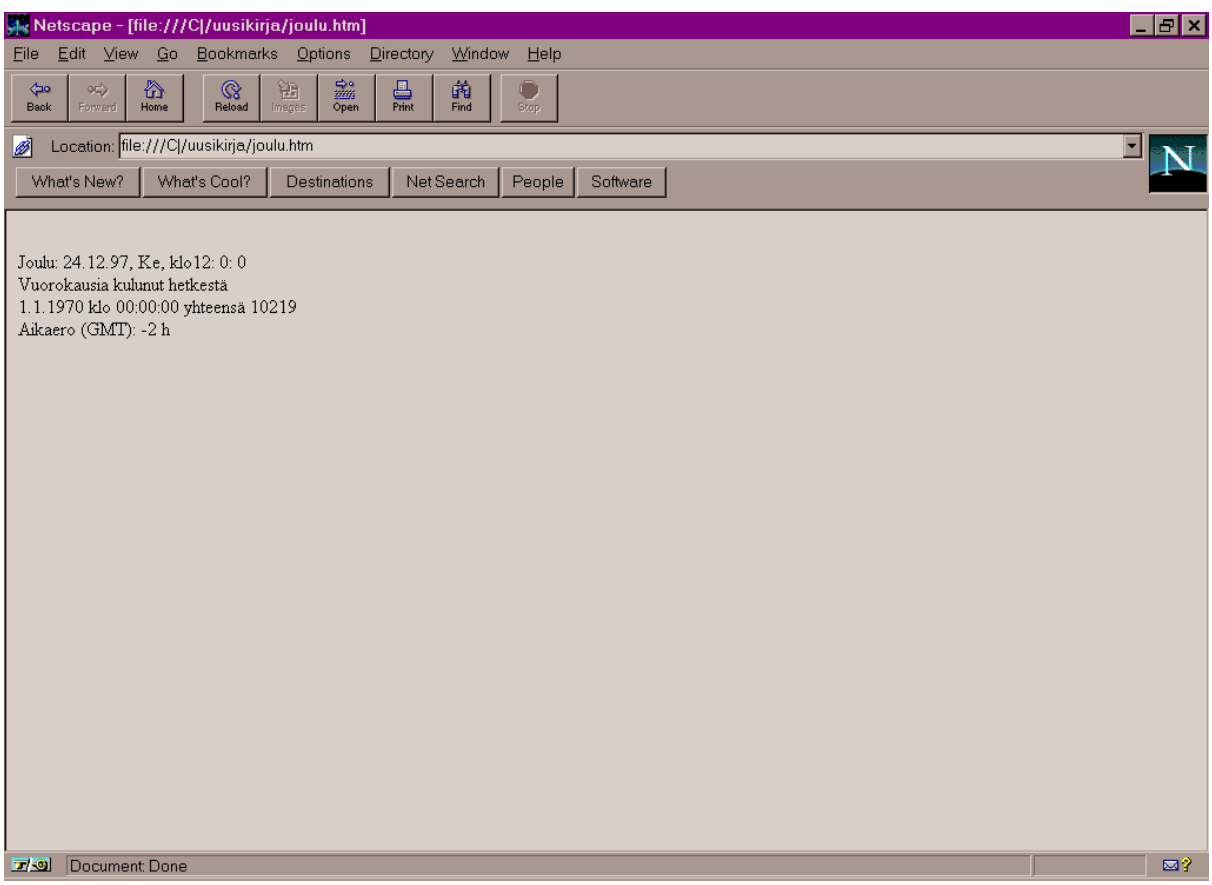
Metodit ovat:

Dateolio.getDate()	Palauttaa kuukauden päivän lukuna väliltä 1-31.
Dateolio.getDay()	Palauttaa viikon päivän lukuna väliltä 0 - 6, missä 0 vastaa sunnuntaita.
Dateolio.getHours()	Palauttaa vuorokauden tunnit lukuna 0 - 23.
Dateolio.getMinutes()	Palauttaa minuutit lukuna 0 -59
Dateolio.getMonth()	Palauttaa kuukaudet lukuna 0 - 11 (nolla edustaa tammikuuta).
Dateolio.getSeconds()	Palauttaa sekunnit lukuna väliltä 0 -59.
Dateolio.getTime()	Palauttaa ajanhetken 1.1.1970 klo 00:00:00 ja nykyhetken välisen ajan millisekunteina.
Dateolio.getYear()	Palauttaa vuoden kaksinumeroisena lukuna.
Dateolio. getTimezoneOffset()	Palauttaa aikaeron minuutteina GMT;n verrattuna.

Esimerkki:

```
        <HTML>
        <BODY>
        <SCRIPT LANGUAGE="JavaScript">

        joulu_97 = new Date("Dec 24, 1997 12:00:00");
        document.write("<BR> Joulu:
",joulu_97.getDate(),".");

        var kk = joulu_97.getMonth() + 1 ;

        document.write(kk,".");
        document.write(joulu_97.getYear(),", ");

        var vkp = joulu_97.getDay();
        var pva = "" ;


        if(vkp == 0)
          pva = "Su" ;

        if(vkp == 1)
          pva = "Ma" ;

        if(vkp == 2)
           pva = "Ti" ;

        if(vkp == 3)
          pva = "Ke" ;

        if(vkp == 4)
          pva = "To" ;

        if(vkp == 5)
          pva = "Pe" ;

        if(vkp == 6)
          pva = "La" ;

        document.write(pva,", ");
        document.write("klo",joulu_97.getHours(),": ");
        document.write(joulu_97.getMinutes(),": ");
        document.write(joulu_97.getSeconds(),"<BR>");

        var mspva = 24*60*60*1000 ;
```

```
         // millisekunteja vuorokaudessa

        var aika_70 = joulu_97.getTime() / mspva ;
        aika_70 = Math.round(aika_70);

        document.write("Vuorokausia kulunut hetkes-
tä<BR>");
        document.write("1.1.1970 klo 00:00:00 yhteensä
",aika_70);

        var aikaero = joulu_97. getTimezoneOffset() / 60
;
        document.write("<BR>Aikaero (GMT): ",aikaero,"
h");

        </SCRIPT>
        </BODY>
        </HTML>
```



Edellisen esimerkin tuloste.

Dateolio.parse(merkkijono) Palauttaa argumenttina olevan

	Date-tyyppisen merkkijonon (Esim. May 1, 1996) millisekunteina ajanhetkestä 1.1.1970 klo 00:00:00.
Dateolio.setDate(arg)	Asettaa argumenttina olevan luvun (luku väliltä 1-31) Dateolion kuukauden päiväksi.
Dateolio.setHours(arg)	Asettaa argumenttina olevan luvun luku väliltä 0-23) Dateolion tunnin arvoksi.
Dateolio.setMinutes(arg)	Asettaa argumenttina olevan luvun (luku väliltä 0-59) Dateolion minuuttien arvoksi.
Dateolio.setMonth(arg)	Asettaa argumenttina olevan luvun (luku väliltä 0-11) Dateolion kuukauden arvoksi.
Dateolio.setSeconds(arg)	Asettaa argumenttina olevan luvun (luku väliltä 0-59) Dateolion sekuntien arvoksi.
Dateolio.setTime(arg)	Asettaa argumenttina olevan ajan (annetaan millisekunteina mitattunaajanhetkestä 1.1.1970 klo 00:00:00) Dateolioon.

Dateolio.setYear(arg) Asettaa argumenttina olevan luvun (luvut väliltä 70-99 tai 2001-2037 tällä hetkellä toimivia) Dateolion vuoden arvoksi.

Dateolio.toGMTString() Muuntaa Date-olioon asetetun ajan GTM ajaksi edellyttäen, että käyttöjärjestelmän aikavyöhyke on asetettu oikein.

Dateolio.toLocaleString() Palauttaa Dateolioon asetetun ajan ‘paikallismuodossa’.

Esimerkki:

```
tanaan = new Date();
document.write("<BR> ",tanaan);
document.write("<BR> ",tanaan.toLocaleString());
document.write("<BR> ",tanaan.toGMTString());
```

Tuottaa tulosteen:

```
Sat Jan 20 16:37:39 1996
01/20/96 16:37:39
Sat, 20 Jan 1996 14:37:39 GMT
```

Date.UTC(vuosi, kk, pva [, h] [, min] [, s])

Palauttaa parametreinä annetun aika ilmaisun millisekunteina ajan hetkestä 1.1.1970 klo 00:00:00. (GMT aikaa).

Parametrit:

- vuosi — luku väliltä 70-99 tai 2001-2037.
- kk — kuukausi, luku väliltä 0-11.
- pva — päivä, luku väliltä 1-31.
- h — tunnit, luku väliltä 0-23.
- min — minuutit, luku väliltä 0-59.
- s — sekunnit, luku väliltä 0-59.

Esimerkki: UTC-metodia pitää käyttää muodossa Date.UTC()

```
paivays = new Date(Date.UTC(96, 0, 21, 0, 0, 0));
document.write("<BR>",paivays);
```

Tuottaa tulosteen:

Sat Jan 20 22:00:00 1996

Esimerkki. Kello.

```
<HTML>
<HEAD>
<!--   Juha Niemi  Last Modified 3.9.1996  -->
<SCRIPT LANGUAGE="javascript">

function aika(apu) {


        paivays=new Date();

        var sek=paivays.getSeconds();

        if(apu ==1) {           // Tahdistus
        var vali=(60 - sek)*1000.1 ; }

        else {
        vali =60001; }
```

```
        var tunnit=paivays.getHours();

        var min=paivays.getMinutes();

        if(min < 10)
        min = "0"+min ;

        var k_aika=tunnit+":"+min;

        document.forms[0].elements[0].value=" "+k_aika+"
";

        viive=setTimeout("vaihda()",vali);

                }

function vaihda() {

        paivays=new Date();

        var tunnit=paivays.getHours();

        var min=paivays.getMinutes();

        if(min < 10)
        min = "0"+min ;

        var k_aika=tunnit+":"+min;

        document.forms[0].elements[0].value=" "+k_aika+"
";

        var apu = 0;

        viive=setTimeout("aika(apu)",60001);

                }


</SCRIPT>
<!--    -->
</HEAD>
<BODY BGCOLOR="#add8e6">
```

```
<FORM NAME="lomio">
<CENTER><TABLE BORDER="12">
<TR><TD>
<INPUT TYPE="text" SIZE="6" NAME="kello" >
</TD></TR></TABLE></CENTER>
</FORM>
<SCRIPT LANGUAGE="javascript">
document.write("<CENTER><B>","J.Niemi's
clock".fontcolor("#000000")),"</B></CENTER>";

var apu=1;
aika(apu);


</SCRIPT>
</BODY>
</HTML>
```

Edellisen esimerkin tuloste ladattuna omaan ikkunaan.

4.8 Form-olio

Form-olio tekee mahdolliseksi merkkijonojen syötön tekstikenttiin (**text**-olio) ja valintojen suorittamisen eri vaihtoehtojen välillä hyödyntäen esimerkiksi **checkbox**-oliota. Form-oliota voidaan käyttää myös datan lähettämiseen palvelimelle. Esimerkiksi lomakkeen sisältö voidaan lähettää cgi-ohjelmalle, joka käsittelee sen ja palauttaa käsittelyn mukaisen tulosteen.

Form-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<FORM
 NAME="olion nimi”
TARGET="ikkunan nimi”"
ACTION="osoite(URL) "
METHOD=GET | POST

ENCTYPE="MIME-tyyppi"    // määrää lähetetyn datan MIME-tyypin

[onSubmit="JavaScript-koodi"]>
</FORM>
```

TARGET

Määrää ikkunan johon palvelimen vastaukset (cgi-ohjelmien tulosteet) tai ladatut tiedostot (dokumentit) tulostetaan.

ACTION

Määrää URL-osoitteen johon syötetty data ohjataan (cgi-ohjelma) tai josta ladataan uusi dokumentti.

METHOD

Määrää miten data lähetetään palvelimelle. GET(oletusarvo) lisää datan URL:ään ja datasta tulee useimmissa palvelimissa ympäristömuuttujan QUERY_STRING arvo. POST lähettää syötetyn datan datalohkossa, joka liitetään stdin;n puskuriin ja sen pituus sijoitetaan ympäristömuuttujaan CONTENT_LENGTH.

Esimerkiksi GET-metodilla lähetetty tieto välittyy **cgi**-ohjelmille parametreina ja POST-metodilla lähetetty tieto voidaan lukea yhtenä merkkijojona **stdin**:stä vaikka **gets**-funktiolla C-kielellä kirjoitettuun **cgi**-ohjelmaan. Vastausohjelman tulosteessa voidaan käyttää myös JavaScriptiä.

Esimerkki: C-kielisen ohjelman tulostusosa, jonka lukuohjelma tulkitsee JavaScriptiksi.

```
printf("Content-type: text/html\n");
printf("\n");
printf("<body><script language='JavaScript'>");
printf("document.bgColor='#FFFFFF';document.fgColor='
#FF0000';");
printf(”document.write(‘<B>Heippa!</B>’)”);
printf("</script></body>");
```

HTML-dokumentin jokaista lomaketta käsitellään erillisenä oliona. Form-olioita voidaan osoittaa koodissa hyödyntämällä **document**-olion muuttujaa **forms**. Forms-muuttuja on taulukkomuuttuja, jolla voidaan osoittaa dokumentin jokaista form-oliota eli lomaketta indeksoidusti. Form-olion elementtejä voidaan osoittaa koodissa käyttämällä form-olion muuttujaa **elements.** Elements-muuttuja on taulukkomuuttuja, jonka avulla voidaan osoittaa form-olion (lomakkeen) elementtejä (**chekbox**-, **radioButton**- , **text**- ym oliot) indeksoidusti.

Muuttujat:

action	Sisältää FORM-tagin ACTION parametrin arvon. Vaikuttaa dynaamisesti
elements	Taulukkomuuttuja, jolla osoitetaan **form**-olion elementtejä. Indeksointi määräytyy tagien kirjoitusjärjestyksen perusteella.

Esimerkki:

```
<FORM NAME=”joku_lomake”>
<INPUT type=”text” name=”naytto”>
```

```
<INPUT type="checkbox" name="valitsin">
</FORM>
```

Joku_lomake nimisen form-olion elementtien, **text**-olio *naytto* ja **checkbox**-olio *valitsin*, muuttujien **value** arvoja voidaan nyt käyttää koodissa seuraavasti. Katso **document**-olio kappaleesta joku_lomake-nimen korvaamisesta **forms**-muuttujalla.

```
document.joku_lomake.elements[0].value    // text-olio naytto
document.joku_lomake.elements[1].value    // checkbox-olio
                                             valitsin
```

encoding Sisältää ENCTYPE-parametrin arvon. encoding-muuttujan arvon muuttaminen vaikuttaa dynaamisesti.

method Sisältää FORM-tagin METHOD parametrin arvon (joko GET tai POST).

name Sisältää FORM-tagin NAME parametrin arvon.

target Sisältää FORM-tagin TARGET parametrin arvon.

Metodi:

submit() Lähettää elementtien sisällön palvelimelle tai aiheuttaa ACTION-parametrilla määrätyn tiedoston lataamisen TARGET-parametrilla määrättyyn ikkunaan (myös cgi-ohjelmien tulosteet ladataan tähän ikkunaan).

Esimerkki: Suoritetaan submit-metodi lomio-nimiselle form-oliolle, joka on dokumentin ensimmäinen form-olio.

```
document.lomio.submit();

tai

document.forms[0].submit();
```

Tapahtumafunktio:

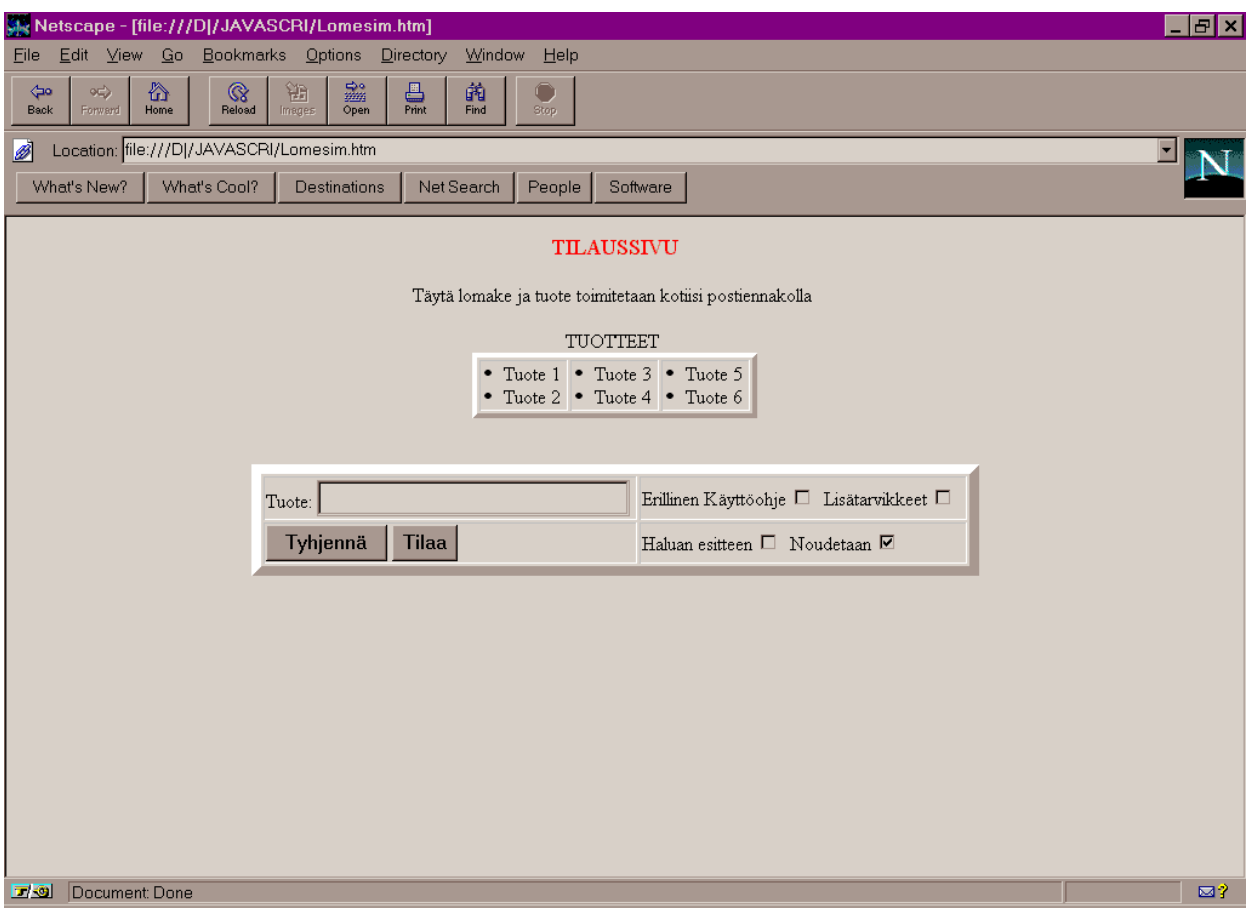
onSubmit	Suoritetaan koodi, joka on liitetty tapahtumafunktioon. Jos paluuarvo on **true**, lomake lähetetään ja jos paluuarvo on **false** lomaketta ei lähetetä.

Esimerkki: Tilauslomake.

```
<HTML>
<BODY>

<SCRIPT LANGUAGE="javascript">

var otsikko = "TILAUSSIVU"

var teksti = "Täytä lomake ja tuote toimitetaan kotiisi
postiennakolla " ;

document.write("<CENTER>",otsikko.fontcolor("#FF0000").bi
g(otsikko),"</CENTER>");

document.write("<CENTER><BR>",teksti.fontcolor("#000000")
,"</CENTER>");

</SCRIPT>
```

```
<BR><CENTER>
TUOTTEET
<TABLE border="4">
<TR>
<TD align="top">
<LI>Tuote 1
<LI>Tuote 2
</TD>

<TD>
<LI>Tuote 3
<LI>Tuote 4
</TD>

<TD>
<LI>Tuote 5
<LI>Tuote 6
</TD>
</TR>
</TABLE>
</CENTER>

<BR>

<FORM NAME="tilaus" METHOD="POST"
ACTION="http://www.kone.fi/cgi-bin/cgi-ohjelma"
onSubmit="if (tilaus.tuote.value.length != 0){return
true;}else {return false;}">

<!-- if-lauseella pyritään estämään tyhjän lomakkeen lä-
hetys -->

<BR>
<CENTER>
<TABLE border="8">
<TR>
<TD>
Tuote:
<INPUT type="text" NAME="tuote" SIZE="30">
</TD>

<TD>
Erillinen Käyttöohje
<INPUT type="checkbox" NAME="listar">

Lisätarvikkeet
<INPUT type="checkbox" NAME="kayohje">
```

```
</TD>
</TR>
<TR><TD>
<INPUT type="reset" VALUE="Tyhjennä">

<INPUT type="submit" VALUE="Tilaa">
</TD>

<TD>
Haluan esitteen
<INPUT type="checkbox" NAME="esite">

Noudetaan
<INPUT type="checkbox" CHECKED NAME="noud">
</TD></TR>
</TABLE>

</CENTER>
</FORM>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

4.9 Form-olion elementit

Form-oliolla on käyttökelpoisia olioita, joita kutsutaan elementeiksi. Monia niistä on jo käytetty **Esimerkki**en yhteydessä. Oliolla **anchor** luodaan hyperlinkin kirjanmerkki dokumentin sisällä. Ankkuri luodaan standardi-**HTML**-kielen avulla.

Esimerkki:

```
<A HREF=”#nimi”>mene tänne</A>
<A NAME="nimi"> täällä</A>
```

“nimi” on hypertekstilinkin osoite.

Muuttuja:

length	sisältää elementtien määrän.

4.9.1 Button-olio

Oliolla **button** saadaan aikaan painonappi. Button-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<INPUT TYPE="button" NAME="olion nimi" VALUE="Painonapin teksti"
  [onClick="JavaScript-koodi"]>
```

NAME-parametrin arvolla osoitetaan painonappia form-olion sisällä.

Muuttujat:

name	Sisältää painonapin nimen.
value	Sisältää painonapin tekstin.

Metodi:

click()	Simuloi hiiren näpäytystä eli tekee painonapista painonapin.

Tapahtumafunktio:

onClick	Suorittaa määrätyn JavaScript-koodin kun nappia painetaan.

Esimerkki. Luodaan painonappi, jonka painalluksesta tulostetaan erilliseen tulostuskehykseen teksti ”*Heippa WWW!*” ja rivinvaihto joka kerta, kun nappia painetaan.

1. Tarvitaan kehyksien määrittelydokumentti (*testframe.htm*).

```
<HTML>
<HEAD>
<TITLE>String-test</TITLE>
</HEAD>

<FRAMESET COLS="50%,*">
        <FRAME SRC="koe.html" NAME="ohjaus">
        <FRAME SRC="tulostus.html" NAME="tulos">
</FRAMESET></HTML>
```

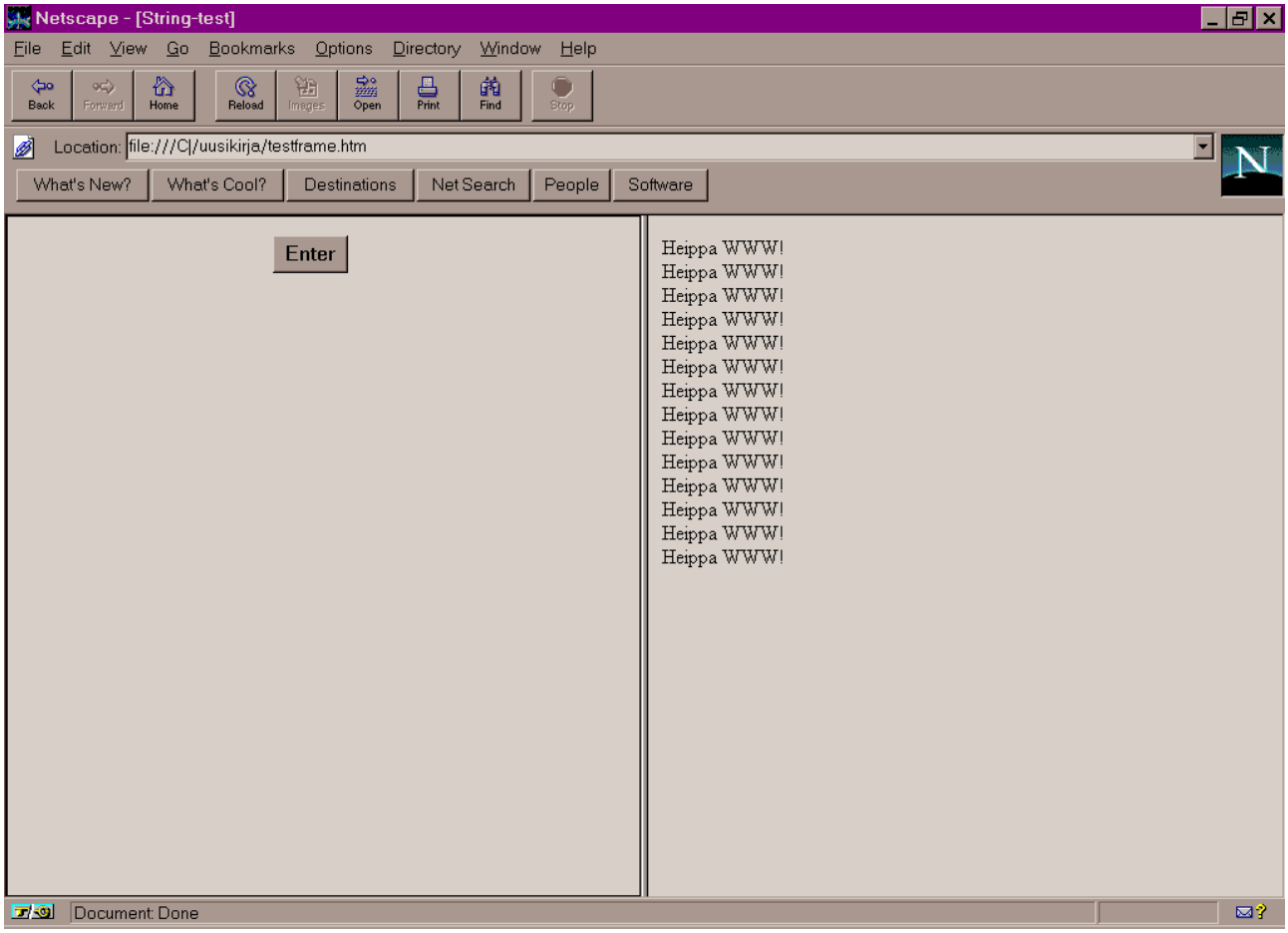
2. Tarvitaan alkusivu tulostuskehykseen (*tulostus.html*).

```
<HTML>
<BODY>

</BODY>
</HTML>
```

3. Tarvitaan varsinainen toimintasivu (*koe.html*).

```
<HTML>
<HEAD>
```

```
<SCRIPT LANGUAGE="JavaScript">

function tulosta(){

parent.tulos.document.write("Heippa WWW!<BR>");

//tulostuskehyksen osoituksesta, katso frame-olio

}
</SCRIPT>
</HEAD>

<BODY>
<CENTER>
<FORM>

<INPUT TYPE="button" NAME="nappi" VALUE="Enter"
onClick="tulosta()">

</FORM>
</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

4.9.2 Checkbox-olio

Oliolla **checkbox** saadaan aikaan rastitettavia ruutuja ja se luodaan standardi-HTML-kielen avulla seuraavasti.

```
<INPUT TYPE="checkbox"   NAME="olion nimi"  [CHECKED]
  [onClick="JavaScript-koodi"]>
```

Jos CHECKED-tunniste on merkitty, tulee ruutu valituksi oletusarvona.

Muuttujat:

checked	Muuttujaan sijoitetaan ruudun tila. Rastitettu = true, tyhjä = false.
defaultChecked	Jos käytetty CHECKED-tunnistetta muuttujan tila = true.
name	Sisältää olion nimen.
value	Sisältää merkkijonon joka on annettu **value**-parametrille arvoksi tai jos mitään ei ole annettu, merkkijonon ”ON” kun ruutu on valittu.

Metodi:

click()	Merkkaa ja poistaa rastin.

Tapahtumafunktio:

onClick	Suorittaa määrätyn JavaScript-koodin kun ruutua napsautetaan hiirellä.

4.9.3 Hidden-olio

Oliolla **hidden** voidaan lomaketta lähetettäessä välittää muutakin kuin käyttäjän syöttämää tietoa. Hidden-olion sisältämä tieto ei ole käyttäjän nähtävissä eikä muutettavissa. Hidden-olio luodaan standardi HTML-kielen avulla seuraavasti.

```
<INPUT TYPE="hidden"  NAME="olion nimi"
  [VALUE="arvo (merkkijono)"]>
```

Muuttujat:

name	Sisältää NAME-parametrin arvon.
value	Sisältää VALUE-parametrin arvon. Vaikuttaa dynaamisesti.

Esimerkki: Tässä otetaan talteen **text**-olion (alku)arvo. Oletetaan, että elementit sijaitsevat dokumentin ensimmäisellä lomakkeella. Text-olio on määritelty kyseisen lomakkeen ensimmäiseksi elementiksi ja **hidden**-olio toiseksi.

```
document.forms[0].elements[1].value  =
document.forms[0].elements[0].value
```

4.9.4 Link-olio

Oliolla **link** saadaan aikaan hypertekstilinkkejä ja se luodaan standardi-HTML-koodin avulla seuraavasti.

```
<A HREF=ankkuri tai URL-osoite TARGET="ikkunan nimi"
[onClick="JavaScript-koodi"] [onMouseOver=
 " JavaScript-koodi "] > linkin teksti </A>
```

TARGET-parametrin arvo määrää ikkunan, johon linkki ladataan. Jokainen **link**-olio on myös **location**-olio.

Muuttuja:

target	Sisältää kohdeikkunan nimen.

Tapahtumafunktiot:

onClick	Lataa linkin.
onMouseOver	Suorittaa määrätyn JavaScript-koodin kun hiiren kohdistin vedetään linkkitekstin päälle.

4.9.5 Password-olio

Password-olion avulla saadaan aikaan tekstikenttä, johon kirjoitettu teksti näkyy asteriskeina. Huomaa, että postitettu teksti näkyy vastaanottajalla selväkielisenä! Password-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<INPUT TYPE="password" NAME="olion nimi" [VALUE="merkkijono"]
SIZE=luku >
```

SIZE-parametrin arvo määrää tekstikentän laajuuden. VALUE-parametrin arvo määrää tekstikentän sisällön.

Muuttujat:

defaultValue	Sisältää tekstikentän oletusarvon.
name	Sisältää olion nimen.
value	Sisältää tekstikentän sisällön merkkijonona.

Metodit:

blur()	Poistaa kursorin tekstikentästä.
focus()	Asettaa kursorin tekstikenttään.
select()	Maalaa (*highlight*) tekstialueen.

4.9.6 Radiobutton-olio

RadioButton-oliolla saadaan aikaan radionappiryhmä, josta valitaan yksi vaihtoehto. RadioButton-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<INPUT TYPE="radio"  NAME="olion nimi" VALUE="merkkijono"
  [CHECKED] [onClick="JavaScript-koodi"] >
```

NAME-parametrin arvon pitää olla sama ryhmän kaikilla napeilla. VALUE-parametrilla määrätään kunkin yksittäisen napin arvo, joka tu-

lee valituksi ryhmästä. CHECKED-parametrilla nappi merkitään valituksi oletusarvona.

Muuttujat:

checked	Sisältää arvon **true,** jos nappia on painettu ja jos ei = **false**.
defaultChecked	Napin oletus arvo: t**rue** = painettu, **false** = ei painettu.
index	Sisältää yksittäisennapin indeksin radionappiryhmässä.
length	Sisältää nappien lukumäärän radionappiryhmässä.
name	Sisältää radionappiryhmän nimen.
value	Sisältää saman merkkijonon kuin VALUE-parametri.

Metodi:

click()	Muuttaa napin tilan.

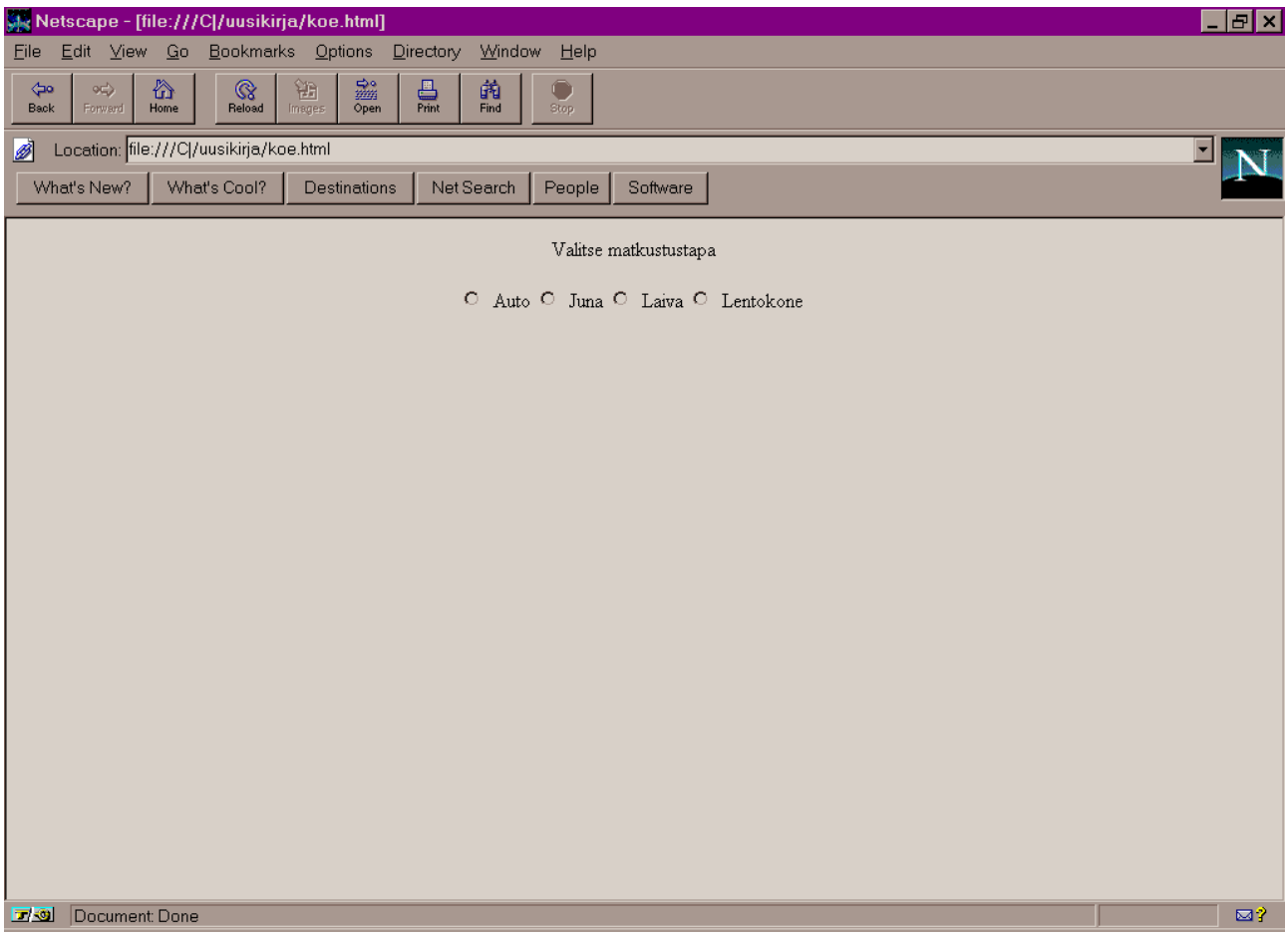
Tapahtumafunktio:

onClick	Suorittaa määrätyn JavaScript-koodin, kun hiirellä napsautetaan nappia.

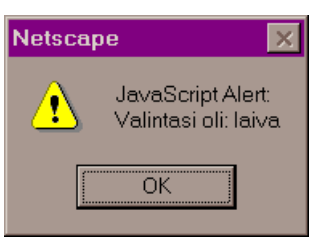
Esimerkki. Neljän napin radionappiryhmä, josta vain yksi valinta on kerrallaan voimassa. Kun valinta tehdään tulostetaan samalla valittu vaihtoehto erilliseen viesti-ikkunaan (**alert()**). Viesti-ikkunasta kerrotaan tarkemmin **window**-olion yhteydessä.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">

function tulosta(syote){

alert("Valintasi oli: " + syote.value);
}

</SCRIPT>
</HEAD>

<BODY>
<CENTER>
Valitse matkustustapa<BR>


<FORM NAME="lomio">

<INPUT TYPE="radio"  NAME="valitsin" VALUE="auto" on-
Click="tulosta(this.form.elements[0]);"> Auto
<INPUT TYPE="radio"  NAME="valitsin" VALUE="juna" on-
Click="tulosta(this.form.elements[1]);"> Juna
<INPUT TYPE="radio"  NAME="valitsin" VALUE="laiva" on-
Click="tulosta(this.form.elements[2]);"> Laiva
<INPUT TYPE="radio"  NAME="valitsin" VALUE="lentokone"
onClick="tulosta(this.form.elements[3]);"> Lentokone

</FORM>
</CENTER>
</BODY>
```

```
</HTML>
```



Edellisen esimerkin tuloste, kun mitään ei ole valittu.



Edellisen esimerkin viesti-ikkuna (tulostuu päällimmäiseksi), kun on valittu laiva.

4.9.7 Reset-olio

Reset-oliolla tyhjennetään lomakkeen tekstikentät tai palautetaan oletusarvot . Reset-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<INPUT  [NAME="olion nimi"]  TYPE="reset"
VALUE="napin teksti"   [onClick="JavaScript-koodi"] >
```

Muuttujat:

name	Sisältää olion nimen.
value	Sisältää painonapin tekstin.

Metodi:

click()	Simuloi hiiren napsautusta.

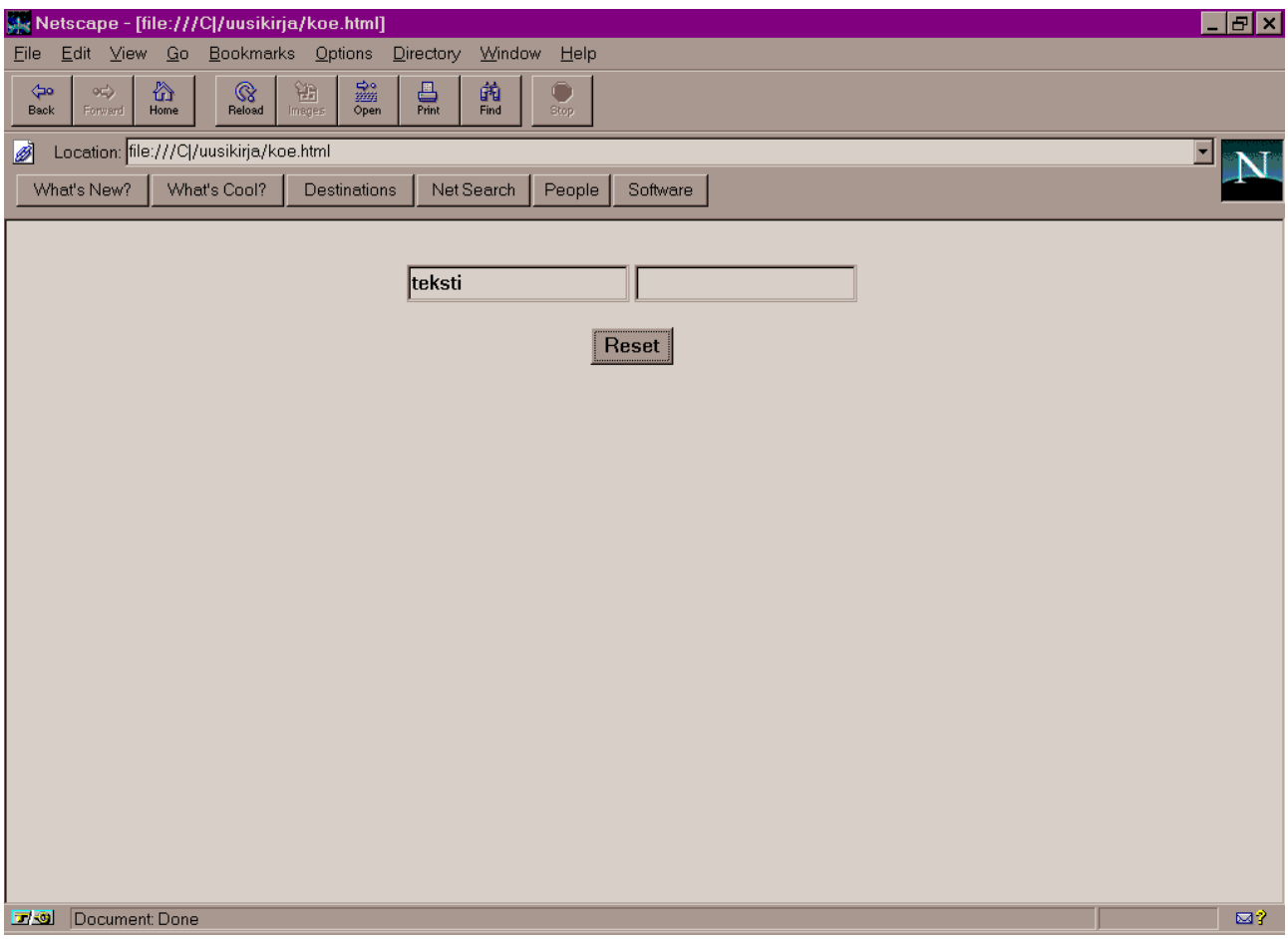
Tapahtumafunktio:

onClick	Suorittaa määrätyn JavaScript-koodin, kun nappia on painettu.

Esimerkki. Ensimmäiseen tekstikenttään annetaan alkuarvo teksti ja toinen jätetään tyhjäksi. Tekstikenttään voidaan kirjoittaa mitä vain ja kun **reset**-nappia painetaan palautuu alkutilanne.

```
<HTML>
<BODY>
<CENTER>
<BR>

<FORM NAME="lomio">

<INPUT TYPE="text"  NAME="eka" VALUE="teksti" >
```

```
<INPUT TYPE="text"  NAME="toka">
<BR><BR>
<INPUT TYPE="reset"  VALUE="Reset">

</FORM>
</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste (alkutilanne).

4.9.8 Selection-olio

Selection-oliolla saadaan aikaan valikkoja. Selection-olio luodaan standardi-**HTML**-kielen avulla seuraavasti.

```
<SELECT NAME="olion nimi"   [SIZE="numero"]   [MULTIPLE]
  [onBlur="JavaScript-koodi"]  [onChange=" JavaScript-koodi "]
  [onFocus=" JavaScript-koodi "] >
  <OPTION [SELECTED]> vaihtoehto_1
  [<OPTION> vaihtoehto_n]
</SELECT>
```

SIZE-parametrin arvo määrää kerralla näkyvien vaihtoehtojen lukumäärän valikossa. MULTIPLE-parametrilla saadaan aikaan valikko, josta voidaan valita useampia vaihtoehtoja.

Muuttujat:

options	Sisältää taulukon, jolla osoitetaan OPTION-parametrilla määrättyjä vaihtoehtoja indeksoidusti.
selectedIndex	Sisältää valitun vaihtoehdon indeksin.

Options-muuttuja sisältää seuraavat muuttujat.

defaultSelected	Sisältää arvon **true**, jos kyseinen vaihtoehto on valittu oletusarvoksi SELECTED-parametrilla, jos oletusarvoa ei ole asetettu = **false**.

index	Sisältää vaihtoehdon (**options**-muuttujan) järjestysnumeron vaihtoehtolistassa alkaen nollasta.
selected	Sisältää arvon **true**, jos kyseinen vaihtoehto on valittu, jos ei = **false**.
text	Sisältää vaihtoehdon merkkijonona.

Esimerkki: **options**-muuttujan ensimmäisen OPTION-parametrin **text**-muuttujaan osoitetaan seuraavasti.

```
select-olion_nimi.options[0].text
```

Metodi:

click()	Asettaa vaihtoehdon valituksi ja poistaa valinnan.

Tapahtumafunktiot:

onBlur	Suorittaa määrätyn JavaScript-koodin kursori poistetaan valikosta.
onChange	Suorittaa määrätyn JavaScript-koodin, kun valittua vaihtoehtoa on muutettu.

onFocus	Suorittaa määrätyn JavaScript-koodin kursori viedään valikkoon.

Esimerkki. Ftp-selain, joka on jaettu kahteen eri kehykseen (*Frames*).

1. Kehysten määrittely.

```
<HTML>
<TITLE>emi's ftp-browser</TITLE>
<FRAMESET COLS="31%,69%" >

<FRAME NAME="toka" SRC="ftp.htm">

<FRAME NAME="kolmas" SRC="ftptex.htm">

</FRAMESET>
</HTML>
```

2. Ftptex.htm, (joka ladataan aluksi toiseen kehykseen).

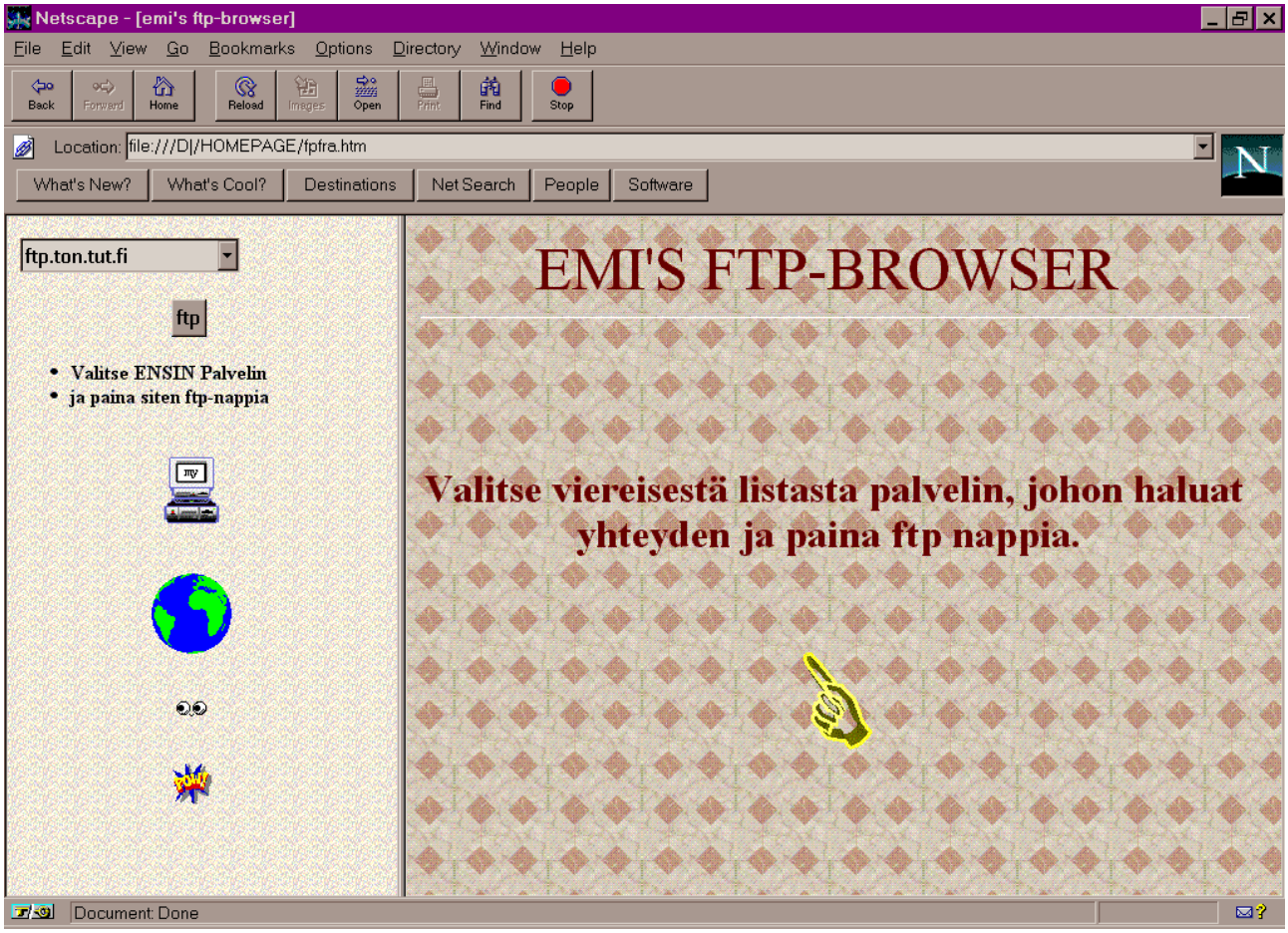
```
<HTML>
<BODY background="cdrback4.gif">
<font color="#5,0,0" size="7">

<CENTER>
EMI'S FTP-BROWSER
<BR><HR><P><BR>

<font size="6"><B>
Valitse viereisestä listasta palvelin, johon ha-
luat yhteyden ja paina ftp nappia.
</B>

<BR><BR><BR>

<IMG SRC="point.gif" border="0">
```

```
        </CENTER>
        </BODY>
        </HTML>
```

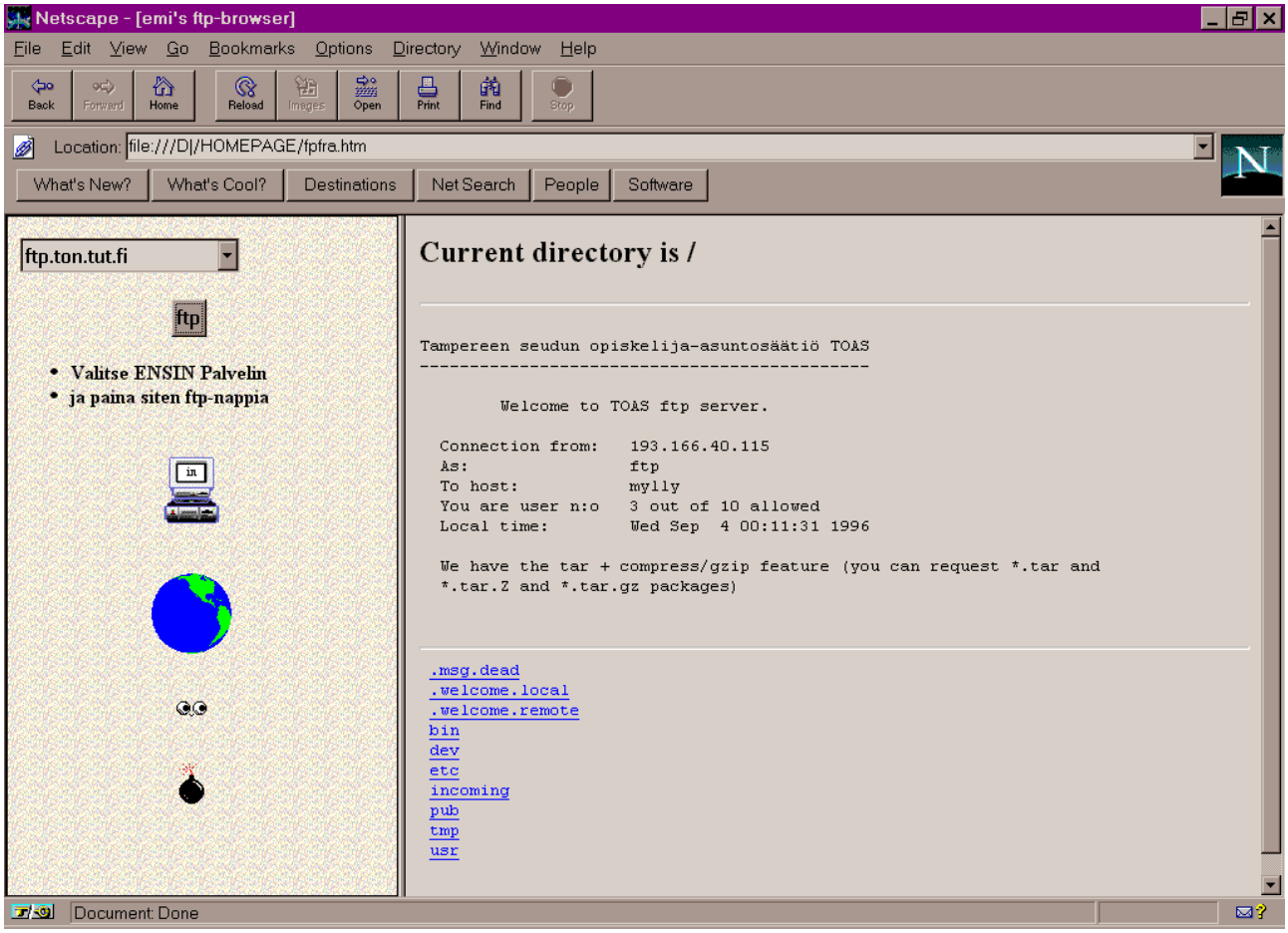
3. Varsinainen selain osa ftp.htm .

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">


        function aseta(syote) {

        var alku="ftp://";

        for(var i=0;i<28;i++)
        {

        if(syote[i].selected)
        document.toinen.action=alku+syote[i].text ;

        }

                        }

</SCRIPT>
</HEAD>
<BODY background="back4.gif">

<FORM  TARGET="kolmas" NAME="toinen">

<SELECT NAME="valikko" SIZE="1" on-
Blur="aseta(this.form.valikko.options)">

<OPTION SELECTED>ftp.ton.tut.fi
<OPTION>ftp.funet.fi<OPTION>ftp.mpoli.fi
<OPTION>ftp.sunet.se<OPTION>ftp.avaks.fi
<OPTION>ftp.eunet.fi<OPTION>ftp.isoc.org
<OPTION>ftp.microsoft.com<OPTION>ftp.NCSA.uiuc.edu
<OPTION>ftp.ed.ac.uk<OPTION>ftp.mcom.com
<OPTION>ftp.einet.net<OPTION>ftp.w3.org
<OPTION>sunsite.unc.edu<OPTION>ftp.cdrom.com
<OPTION>ftp.spry.com<OPTION>ftp.uni-koeln.de
<OPTION>oak.oakland.edu<OPTION>ftp.netscape.com
```

```
<OPTION>ftp2.netscape.com<OPTION>ftp3.netscape.com
<OPTION>ftp4.netscape.com<OPTION>ftp5.netscape.com
<OPTION>ftp6.netscape.com<OPTION>ftp7.netscape.com
<OPTION>ftp.cs.umn.edu<OPTION>server.berkeley.edu
<OPTION>ftp.leo.chubu.ac.jp
</SELECT>
<BR><BR><CENTER>
<INPUT type="submit" value="ftp">
</CENTER>
</FORM>

<UL><B>
<LI>Valitse ENSIN Palvelin
<LI>ja paina siten ftp-nappia
</UL><CENTER></B>

<BR>
<IMG SRC="computer.gif" border="0"><BR><BR><BR>

<IMG SRC="anearth.gif" border="0"><BR><BR><BR>

<IMG SRC="eyescan.gif" border="0"><BR><BR><BR>

<IMG SRC="bombpow.gif" border="0"><BR><BR><BR>
<!--
<A HREF="http://impedanssi.spt.fi:8998/cgi-bin/heippa"
TARGET="kolmas" onMouseOver="window.status='Kuka olet? -
mistä tulet?';return true;">
<IMG SRC="konelin.jpg" border="0"></A>
-->


</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste, kun ftp-yhteyttä ei ole avattu.



Edellisen esimerkin tuloste, kun ftp-yhteys on saatu. Seuraamalla linkkejä oikeassa kehyksessä, voidaan tiedostoja imuroida.

4.9.9 Submit-olio

Submit-oliolla lähetetään lomake ACTION-parametrilla määriteltyyn osoitteeseen. Submit-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<INPUT TYPE="submit"  NAME="olion nimi"  VALUE="painonapin teksti"
  [onClick="JavaScript-koodi"]>
```

Muuttujat:

name	Sisältää olion nimen.
value	Sisältää painonapin tekstin.

Metodi:

click	Simuloi painonapin painallusta.

Tapahtumafunktio:

onClick	Suorittaa määrätyn JavaScript-koodin, kun nappia on painettu.

4.9.10 Text-olio

Text-oliolla saadaan aikaan tekstin syöttö- ja näyttökenttiä. Text-olio luodaan standardi-HTML-kielen avulla seuraavasti. Kentän sisältöä voidaan dynaamisesti päivittää.

```
<INPUT TYPE="text"   NAME="olion nimi"
 VALUE="kentän sisältö  merkkijonona"
 SIZE=luku   [onBlur="JavaScript-koodi"]
[onChange="JavaScript-koodi"]   [onFocus="JavaScript-koodi”]
[onSelect="JavaScript-koodi"]>
```

SIZE-parametrilla määräätään kentän koko.

Muuttujat:

defaultValue	Sisältää kentän alkuarvon. Alkuarvo voidaan antaa VALUE-parametrillä.
name	Sisältää text-olion nimen.
value	Sisältää kentän sisällön.

Metodit:

blur()	Simuloi **onBlur**-tapahtumafunktiota.
focus()	Simuloi **onFocus**-tapahtumafunktiota.
select()	Simuloi **onSelect**-tapahtumafunktiota.

Tapahtumafunktiot:

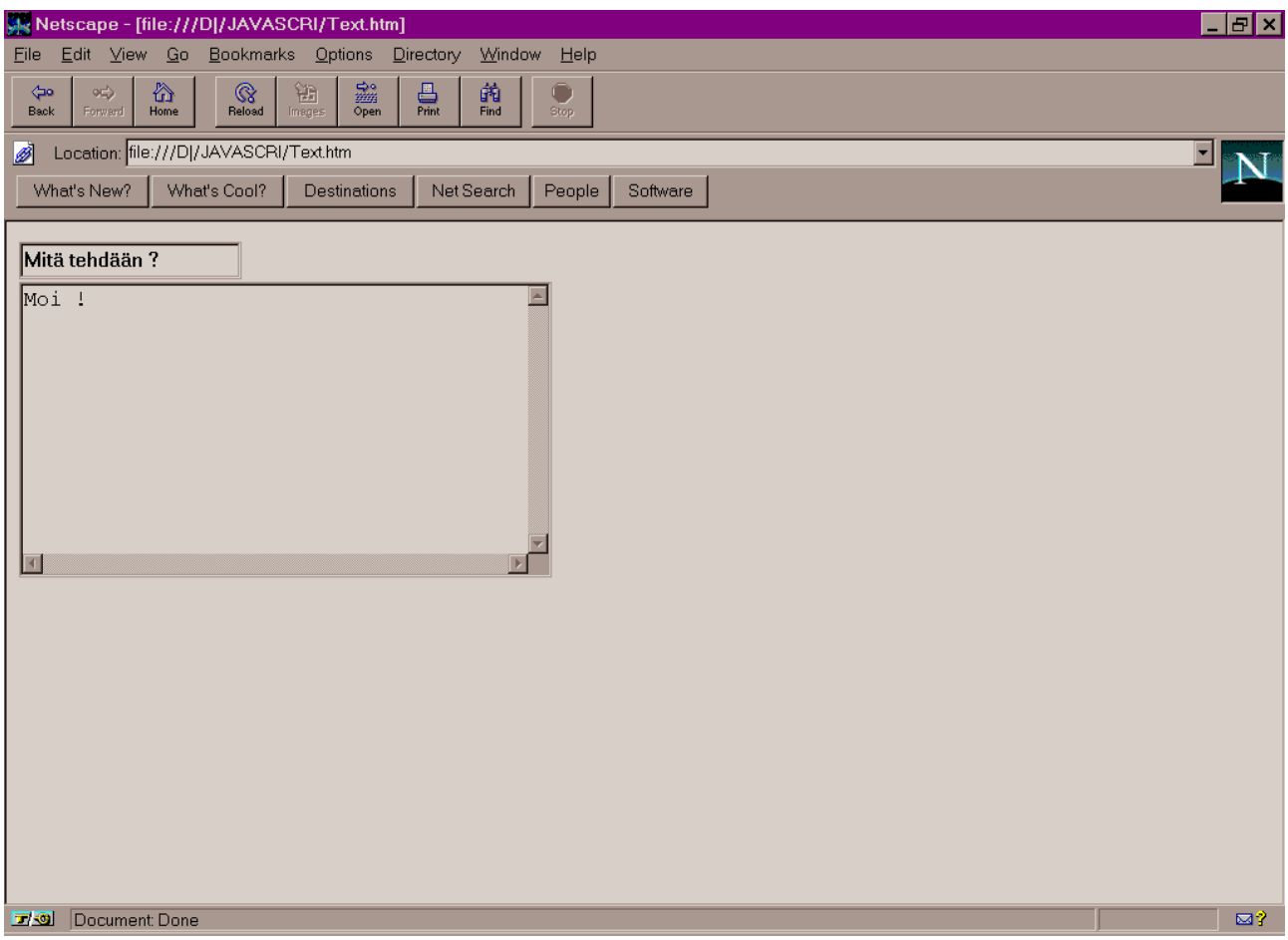
onBlur	Suorittaa määrätyn JavaScript-koodin ,kun kursori siirretään pois text-olion kentästä.
onChange	Suorittaa määrätyn JavaScript-

	koodin, kun text-olion kentän sisältöä on muutettu.
onFocus	Suorittaa määrätyn JavaScript-koodin, kun kursori siirretään text-olion kenttään.
onSelect	Suorittaa määrätyn JavaScript-koodin, kun text-olion kenttä on maalattu.

Esimerkki: **Text**-olion kentän sisältö muuttuu dynaamisesti kun kursori viedään kenttään (“Mitä tehdään ?”) ja siirretään pois sieltä (“Moi !”). Alkuarvoksi annettu “Moi !”.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">

function pois(syote)
{
   syote.value = "Moi !";
}


    function kursori(syote)
    {
      syote.value = "Mitä tehdään ?";
    }

</SCRIPT>
</HEAD>
<BODY>

<FORM NAME="koelomio">

<INPUT TYPE="text"    NAME="naytto"    VALUE="Moi !"
SIZE="20"
```

```
onBlur="pois(this.form.naytto)"

onFocus="kursori(this.form.naytto)">

</FORM>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

4.9.11 Textarea

Textarea-oliolla saadaan aikaan tekstialueita. Tekstialueellle voidaan sijoittaa ainoastaan ASCII-muotoista tekstiä. Textarea-olio luodaan standardi-HTML-kielen avulla seuraavasti.

```
<TEXTAREA  NAME="olion nimi"   ROWS="luku"  COLS="luku"
[onBlur="JavaScript-koodi"]    [onChange="JavaScript-koodi"]
[onFocus="JavaScript-koodi"]   [onSelect="JavaScript-koodi"]>

  //Alueen teksti tähän.
</TEXTAREA>
```

Parametreillä ROWS ja COLS määrätään alueen koko pysty- ja vaakariveinä. Tekstialueella rivinvaihdot huomioidaan. Textarea-oliolla on samat muuttujat ja metodit sekä tapahtumafunktiot kuin text-oliolla.

4.10 Frame-olio

Frame-oliolla saadaan aikaan dokumenttiin useita itsenäisiä kehysikkunoita. Jokaiseen kehykseen voidaan ladata eri URL. Frame-olio luodaan standardi HTML-kielen avulla seuraavasti.

```
<FRAMESET
   ROWS="rivien määrä"                 // Prosentteina tai pikseleinä
   COLS="pystyrivien määrä"            // Prosentteina tai pikseleinä
   [onLoad="JavaScript-koodi"]
   [onUnload="JavaScript-koodi"]>
   [<FRAME SRC="URL" NAME="Kehyksen nimi">]
</FRAMESET>
```

Muuttujat:

frames	Sisältää taulukon, jolla osoitetaan indeksoidusti pääikkunan “lapsi-ikkunoihin” (<FRAMESET> … </FRAMESET>).
name	Sisältää FRAME-tagin NAME-parametrin arvon.
length	Sisältää lapsi-ikkunoiden lukumäärän.
parent	Synonyymi ikkunalle tai pääkehykselle.
self	Synonyymi nykyiselle kehykselle.
window	Kuten **self**-muuttuja.

Metodit:

tunnus=**setTimeout(**lause, aika**)**

Lause (annetaan merkkijonona) suoritetaan kun **millisekunteina** annettu aika on kulunut. **Tunnusta** käytetään peruuttamaan toiminto **clearTimeout()** metodin avulla.

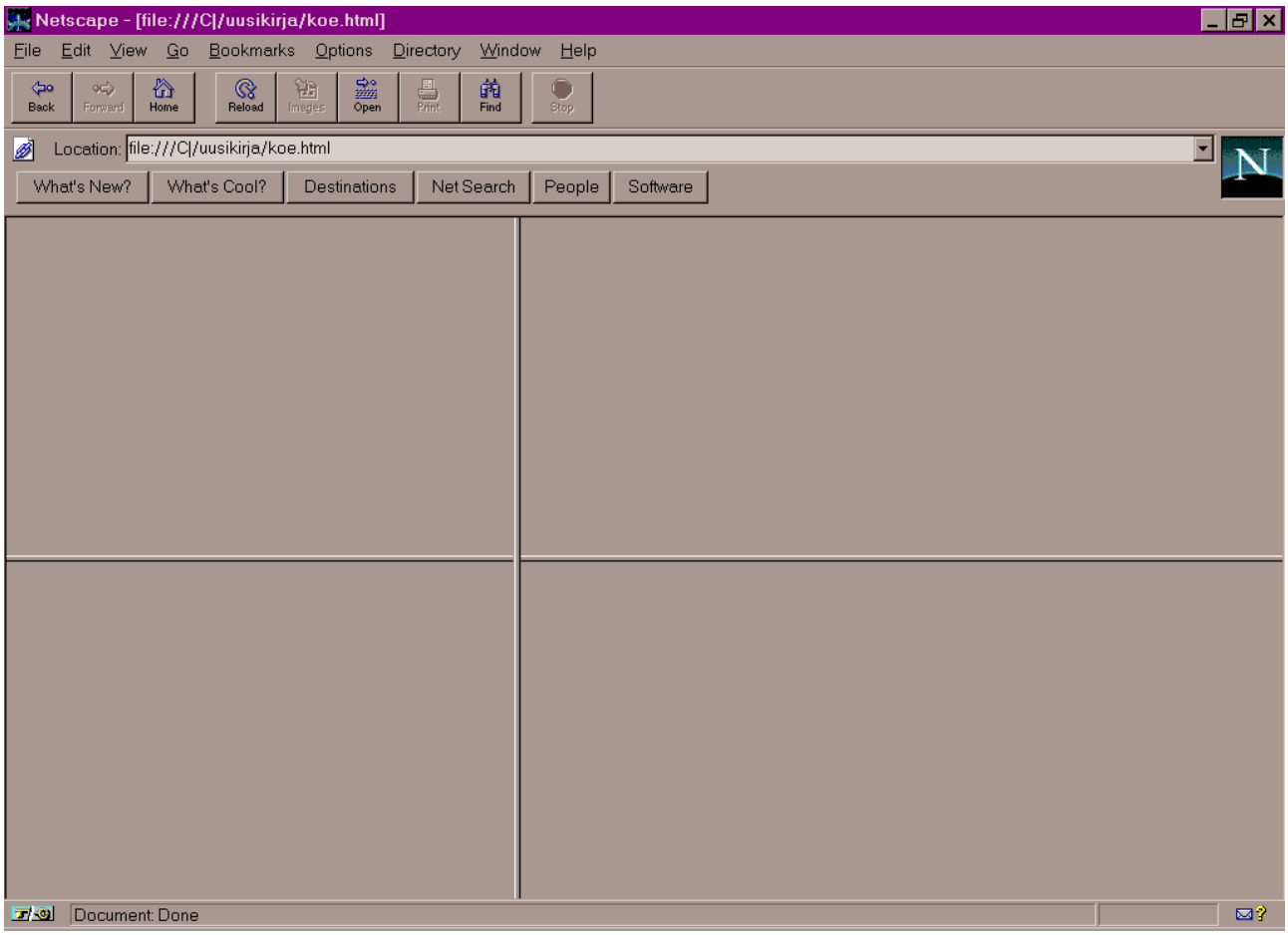
clearTimeout(tunnus**)** Nollaa sen **setTimeout()**-metodin, jonka annettu **tunnus** vastaa sulkeissa olevaa.

Tapahtumafunktiot:

onLoad ja **onUnload**	Molemmat tapahtumafunktiot ovat itse asiassa **window**-olion Tapahtumafunktiota vaikka ne voidaan määritellä <FRAMESET> -tagissa.

Esimerkki: Luodaan sivu, jossa on neljä kehysikkunaa, joihin jokaiseen ladataan oma HTML-dokumentti. Vaaka- ja pystyrivit voidaan antaa joko lukuina tai prosentteina. Lisäksi latauksen yhteydessä näytetään viesti-ikkuna.

```
<HTML>
<FRAMESET ROWS="50%,50%" COLS="40%,60%"
   onLoad="alert('Hello, World.')">
<FRAME SRC=”http://polku/joku1.html” NAME="kehys1">
<FRAME SRC= ”http://polku/joku2.html NAME="kehys2">
<FRAME SRC=”http://polku/joku3.html NAME="kehys3">
<FRAME SRC= ”http://polku/joku4.html NAME="kehys4">
</FRAMESET>
</HTML>
```



Edellisen esimerkin tuloste. (Ainoastaan ikkunan jako kehyksiin on näkyvissä, koska varsinaisia kehyksiin ladattavia dokumentteja ei ollut).

4.11 Navigaattori-olio

Navigator-oliota käytetään selvittämään käyttäjän selainohjelman versiota (älä sotke yleistä navigaattori-oliot ryhmää tähän). Navigator-olion syntaksi on seuraava.

```
navigator.muuttuja
```

Muuttujat:

appCodeName Sisältää käytetyn selainohjelman koodinimen.

Esimerkki: Jos käytössä on Netscape 3.0 saadaan seuraavasta tulosteeksi Mozilla.

```
document.write(navigator.appCodeName);
```

appName Sisältää selainohjelman nimen. Edellisessä esimerkissä **navigator.appName** = Netscape.

appVersion Sisältää versioinformaation muodossa versio (käyttöliittymä,maa). Maa joko I = kansainvälinen tai U = USA.

Esimerkki. Jos käytössä on Netscape 3.0 beta 5 windows-versio niin seuraavasta saadaan tulosteeksi 3.0b5 (Win95, I).

```
document.write(navigator.appVersion);
```

userAgent Sisältää header-tiedoston arvon, jonka asiakasohjelma lähettää palvelinohjelmalle, jotta tämä tunnistaisi sen. Edellisen esimerkin vastaava arvo olisi Mozilla/3.0b5 (Win32; I).

4.12 Yleiset Navigaattori-oliot

Selain-ohjelma luo automaattisesti niin sanottuja navigaattori-olioita sivun latauksen yhteydessä. Seuraavat navigaattori-oliot luodaan jokaiselle sivulle.

- **window**
- **location**
- **history**
- **document**

Navigaattori-oliot noudattavat tiettyä hierarkiaa. Tässä hierarkiassa olion "jälkeläiset" ovat olion itsensä muuttujia. Esimerkiksi **form**-olio on **document**-olion muuttuja.

Tietyn olion tiettyyn muuttujaan osoitettaessa pitää koko "perillis-ketju" olla mukana, poikkeuksena window-olio. Esimerkki. osoitetaan nimetyn **form**-olion (lomio) elementin **text**-olio (nimi = teksti) **value** muuttujan sisältöä.

```
document.lomio.teksti.value
```

4.12.1 Document-olio

Document-olio sisältää koko HTML-dokumentin informaatíon. Selain-ohjelma luo automaattisesti document-olion kyseiselle sivulle, kun sivu latautuu muistiin. Jokaista sivua kohti on vain yksi document-olio. Document-olio voidaan määritellä halutun kaltaiseksi standardi-HTML-kielen avulla seuraavasti.

```
<BODY  BACKGROUND="Taustakuva(URL)"
  BGCOLOR="#Taustan väri "        //  Esim.  "#FF0000"
  FGCOLOR="#Tekstin väri "        //  Esim.  "#00FF00"
```

```
LINK="#Seuraamattoman linkin väri"  //  Esim.  “#0000FF”

ALINK="#Aktiivisen linkin väri" //  Esim.  “#00FFFF”
VLINK="#Seuratun linkin väri" //  Esim.  “#00AA00”

  [onLoad="JavaScript-koodi"]
  [onUnLoad="JavaScript-koodi"]>

</BODY>
```

Muuttujat:

alinkColor	Sisältää ALINK-parametrin arvon. Vaikuttaa dynaamisesti.
anchors	Sisältää taulukon, joka osoittaa indeksoidusti dokumentin jokaiseen **anchor**-olioon.

Esimerkki: Osoitetaan dokumentin ensimmäiseen anchor-olioon.

```
document.anchors[0]
```

bgColor	Sisältää BGCOLOR-parametrin arvon. Vaikuttaa dynaamisesti.
cookie	Sisältää cookie:n sisällön merkkijonona. (cookie on mekanismi, jonka avulla CGI-ohjelmat voivat noutaa tai varastoida tietoa asiakasohjelmaan).

Tarkempaa tietoa cookie-määrittelystä kappaleessa Cookie.

fgColor	Sisältää FGCOLOR-parametrin arvon.
forms	Sisältää taulukon nimettyihin form-olioihin (lomakkeisiin), joita voidaan osoittaa indeksoidusti.

Esimerkki: Osoitetaan ensimmäisen lomakkeen elementtiin, joka on nimetty napiksi.

```
document.forms[0].nappi
```

tai

```
document.forms[0].elements[0]
```

Vastaavasti osoitetaan nappi-olion muuttujaan value.

```
document.forms[0].nappi.value
```

tai

```
document.forms[0].elements[0].value
```

Jälkimmäinen tapa on useimmiten parempi, koska napiksi nimettyä oliota ei välttämättä tunnisteta koko dokumentin alueella

lastModified	Sisältää viimeisen muutoshetken merkkijonona.
linkColor	Sisältää LINK-tunnisteen arvon. Vaikuttaa dynaamisesti.

links	Sisältää taulukon, jolla osoitetaan dokumentinlinkkeihin (**link**-olio) indeksoidusti.
location	Sisältää dokumentin täydellisen URL-osoitteen.
referrer	Sisältää lähtödokumentin URL:n , kun käyttäjä napsauttaa linkkiä hiirellä. Sen arvo voidaan lukea määränpäädokumentissa.
title	Sisältää dokumentin otsikon <TITLE>Otsikko </TITLE>.
vlinkColor	Sisältää VLINK-tunnisteen arvon. Vaikuttaa dynaamisesti.

Metodit:

clear()	Tyhjentää ikkunan.
close()	Sulkee open()-metodilla avatun tietovirran ja tulostaa sen.
open(["MIME-tyyppi"]**)**	Avaa tietovirran, johon ohjataan write()- ja writeln()-metodien tuloste.

MIME-tyyppi voi olla joku seuraavista:

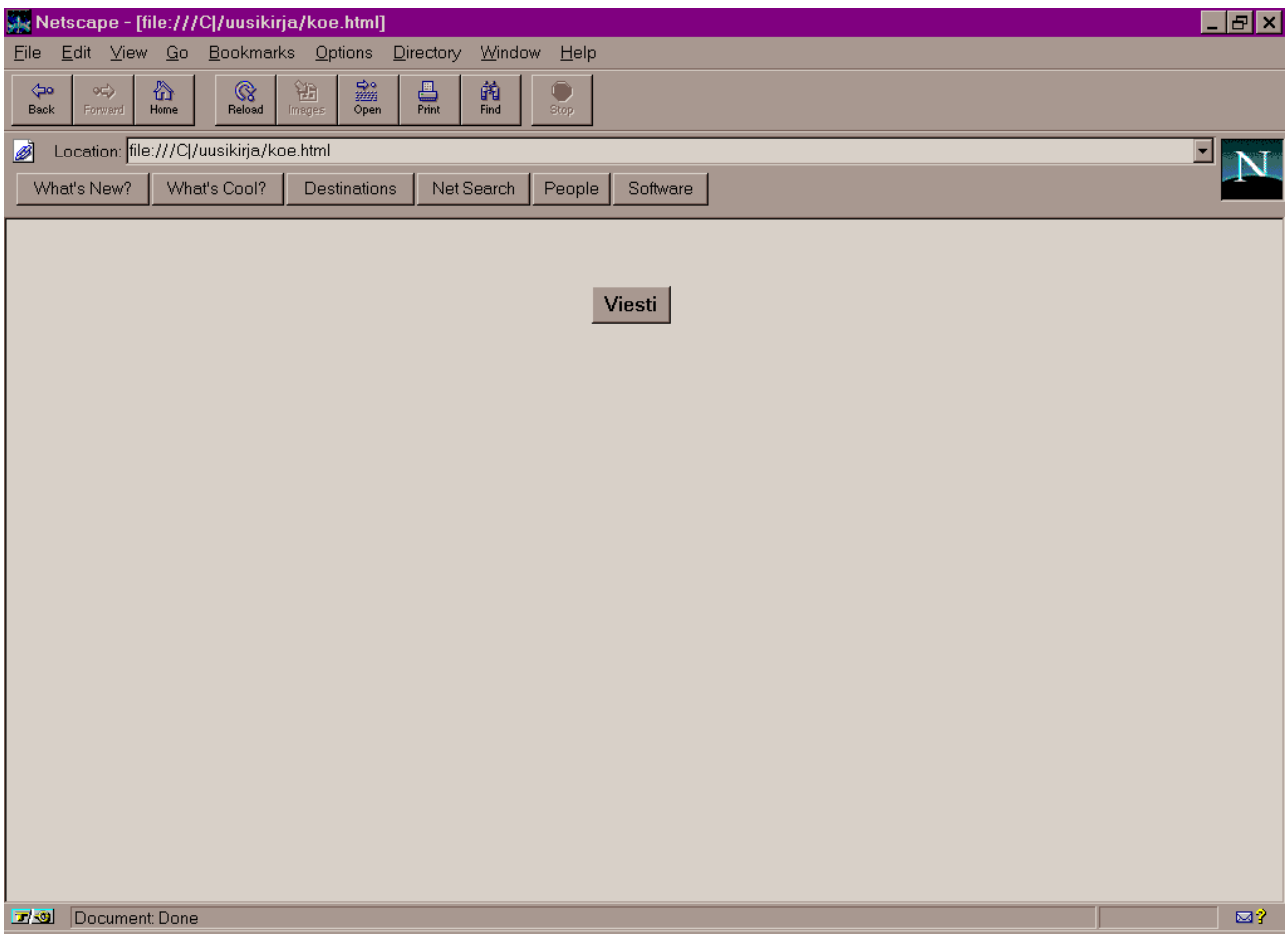
- text/html // *Teksti HTML*
- text/plain // *Teksti tavallinen*
- image/gif // *Kuva gif*
- image/jpeg // *Kuva jpeg*
- image/xbm // *Kuva xbm*
- x-world/plugIn // *Mikä tahansa plug-in MIME-tyyppi, jota netscape tukee. Esimerkiksi x-world/vrml lataa VR Scout VRML plug-in-ohjelman.*

Jos MIME-tyyppi on teksti tai kuva niin tietovirta avataan normaalisti ikkunan dokumenttiin. Muulloin avataan sopiva **plug-in** ohjelma, johon tietovirta ohjataan. Jos dokumentti on jo olemassa niin **open()**-metodi tyhjentää sen ensin. MIME-tyyppi on valinnainen argumentti, joka määrää dokumentin tyypin johon kirjoitetaan. Oletusarvona on **text/html** jos tyyppiä ei määrätä.

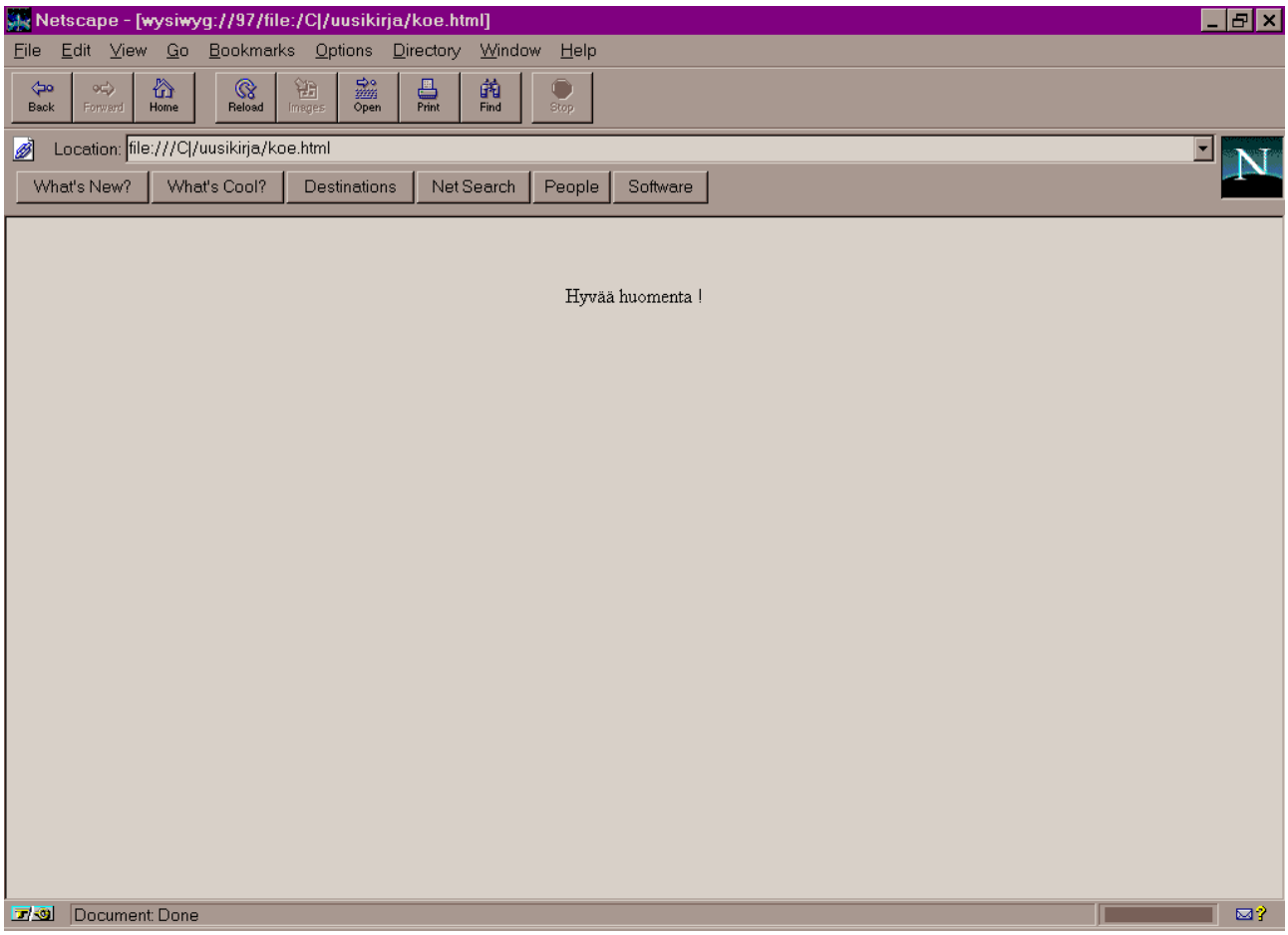
Esimerkki: Seuraava funktio tyhjentää dokumentin ja tulostaa HTML-tekstiä.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">

function viesti() {
        var teksti = "<CENTER>Hyvää huomenta !</CENTER>";
        document.open();
        document.write("<BR><BR>",teksti);
        document.close();
                }

</SCRIPT>
</HEAD>

<BODY>
<CENTER>

<FORM NAME="lomio">
```

```
<BR><BR>
<INPUT TYPE="button"  VALUE="Viesti" onClick="viesti();">

</FORM>
</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste kun viesti-nappia ei ole painettu.



Edellisen esimerkin tuloste kun viesti-nappia on painettu.

---

write(tuloste**)**	Tulostaa argumenttina annettavan muuttujan tai merkkijonon.

Esimerkki: Tulostetaan yhdistetty merkkijono Tämä on koe.

```
document.write(“Tämä” + “ on” + “ koe”);
```

Esimerkki: Tulostetaan kukin merkkijono (myös tyhjät) erikseen.

```
document.write("Tämä", " ", "on", " ",  " koe");
```

writeln(tuloste**)**	Kuten **write()** metodi, mutta lisää rivinvaihtomerkin joka tulosteen jälkeen. Vaikuttaa vain <PRE> … </PRE> tagien kanssa.

Tapahtumafunktiot:

onLoad	Suorittaa määrätyn JavaScript-koodin, kun sivu on ladattu.
onUnLoad	Suorittaa määrätyn JavaScript-koodin, kun sivulta poistutaan.

4.12.2 History-olio

History-olio sisältää URL-informaation sivuista, joilla on vierailtu. Sitä käytetään seuraavasti.

```
history.metodi()
```

tai

```
history.muuttuja
```

Muuttujat:

current	Sisältää nykyisen sivun URL-osoitteen.
length	Sisältää URL-osoitteiden lukumäärän historialistassa.

Metodit:

back()	Lataa edellisen sivun historialistasta. Vastaa metodia **go(-1)**.
forward()	Lataa seuraavan sivun historialistasta. Vastaa metodia **go(1)**.
go(luku | merkkijono**)**	Lataa sivun, joka on luvun osoittaman määrän eteenpäin (luku positiivinen) tai taaksepäin (luku negatiivinen) listassa. Merkkijonoa käytettäessä lataa viimeisimmän sivun jonka otsikkoon tai URL-osoitteeseen (tai niiden osaan) merkkijono täsmää.

Esimerkki: Siirrytään historialistassa 5 sivua taaksepäin.

```
history.go(-5);
```

4.12.3 Location-olio

Location-olio sisältää informaation nykyisestä sivusta. Sitä käytetään seuraavasti.

```
location.muuttuja

tai

location.metodi
```

Muuttujat:

hash	Sisältää nykyisen ankkurin nimen.
host	Sisältää URL-osoitteen käyttämän palvelimen ja portin eroteltuna kaksoispisteellä. Jos porttiosa = null niin sisältö on sama kuin **hostname**-muuttujalla.
hostname	Sisältää URL-osoitteen palvelimen nimen.
href	Sisältää koko URL-osoitteen merkkijonona.
pathname	Sisältää URL-osoitteen polkuosan.
port	Sisältää URL-osoitteen

	protokollan käyttämän portin numeron, jos ei käytössä = “”.
protocol	Sisältää URL-osoitteen protokollaosan kaksoispisteineen.
search	Sisältää kaikki merkit luku ohjelman URL(Location)-kentässä ?-merkin jälkeen. (cgi-ohjelmat saavat tämän merkkijonon parametrina).

4.12.4 Window-olio

Window-olio on kaikkien sivun olioiden “esi-isä”. Window-olion muuttujia ja metodeita voidaan käyttää ilman, että käskyissä esiintyy olio-osaa. Poikkeuksena tilanteet, joissa sen metodit voidaan sekoittaa **document**-olion metodien kanssa. Esimerkiksi seuraava käsky sulkee nykyisen ikkunan eli koko ohjelman, jos muita ikkunoita ei ole avattu.

```
close();
```

Muuttujat:

frames	Sisältää taulukon, jolla osoitetaan indeksoidusti pääikkunan “lapsi-ikkunoihin” (<FRAMESET> … </FRAMESET>).

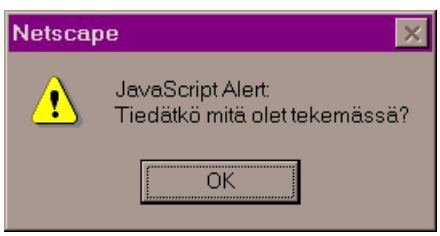
parent	<FRAMESET> ja <FRAME> yhteydessä <FRAMESET> ikkuna.
self	Osoittaa nykyiseen ikkunaan. **Self.close()** , **window.close()** ja **close()** vastaavat toisiaan.
top	Ensimmäinen avattu ikkuna.
status	Sisältää status-kentän sisällön ikkunan alareunassa. Muutokset vaikuttavat dynaamisesti.
defaultStatus	Sisältää status-kentän oletussisällön (sisältö, kun ‘mitään ei tapahdu’).

Metodit:

alert(viesti**)**	Näyttää viesti-ikkunan, joka kuitataan OK-painonapilla. Viesti voi olla mikä tahansa merkkijono.

Esimerkki.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE=”JavaScript”>

var ok=alert(”Tiedätkö mitä olet tekemässä?”);

</SCRIPT>
</BODY>
</HTML>
```



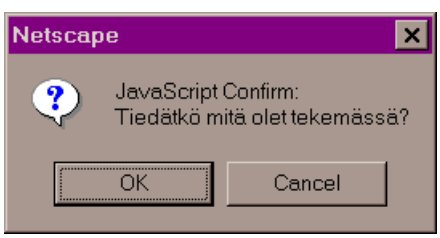
Esimerkin tuloste.

---

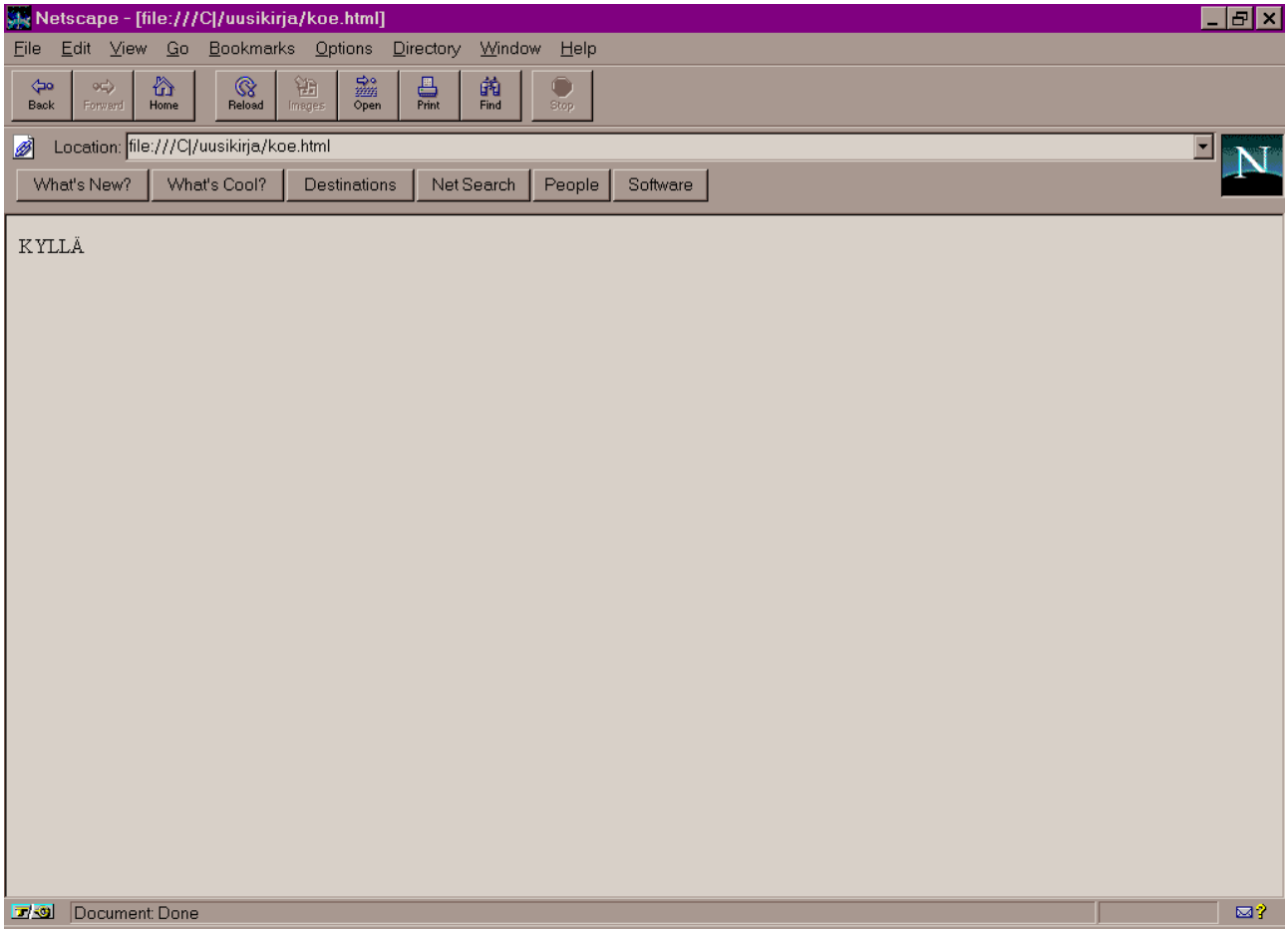
close()	Sulkee nykyisen ikkunan. Jos ikkuna on ainoa auki oleva niin koko selainohjelma suljetaan.
confirm(viesti)	Näyttää viesti-ikkunan, josta valitaan joko OK tai CANCEL. Confirm-metodi palauttaa **true**, jos on valittu OK ja **false** jos on valittu CANCEL.

Esimerkki.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">

var tulos= "EN";

var ok=confirm("Tiedätkö mitä olet tekemässä?");

if(ok)
tulos = "KYLLÄ";

document.write(tulos);

</SCRIPT>
</BODY>
</HTML>
```



Esimerkin tuloste sivulle tultaessa.



Esimerkin tuloste, kun OK-nappia on painettu.

---

open("URL-osoite","ikkunan nimi",["ikkunan ominaisuudet"])

Avaa uuden ikkunan johon ladataan URL-osoite jos sellainen on annettu, muuten avataan tyhjä ikkuna. **Sulkeiden sisällä ei sallita välilyöntejä**. Open-metodin yhteydessä on käytettävä window.open() ilmaisua, jotta se ei sekaantuisi document.open() ilmaisun kanssa. Ikkunan ominaisuudet ovat seuraavat.

```
toolbar[=yes|no]|[=1|0]          // työkalurivi
location[=yes|no]|[=1|0]         // osoitekenttä
directories[=yes|no]|[=1|0]      // hakemistot
status[=yes|no]|[=1|0]           // status-kenttä
menubar[=yes|no]|[=1|0]          // valikkorivi
scrollbars[=yes|no]|[=1|0]       // vierityspalkit
```

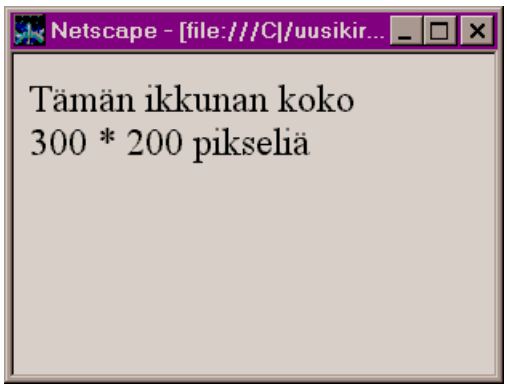
```
resizable[=yes|no]|[=1|0]          // koko muutettavissa
copyhistory[=yes|no]|[=1|0]        // historia
width=pixels                       // leveys pikseleinä
height=pixels                      // korkeus pikseleinä
```

Jos ikkunan nimi ei viittaa olemassa olevaan ikkunaan ja ikkunan ominaisuuksia ei määritellä on kaikki ominaisuudet voimassa oletusarvona. Jos ikkunaan halutaan osoittaa koodissa myöhemmin esimerkiksi document.write-metodilla niin ikkuna on avaamisen yhteydessä sijoitettava muuttujaan. Esimerkiksi.

```
viesti=window.open(””,'viesti');
viesti.document.write(”Kahvitauko !”);
```

Esimerkki.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">

var tulos= "EN";

var ok=confirm("Tiedätkö mitä olet tekemässä?");

if(ok)
tulos = "KYLLÄ";

if(tulos=="KYLLÄ")

ikkuna=window.open('koko.html','ikkuna','toolbar=no,locat
ion=no,directories=no,menubar=no,status=no,copyhistory=no
,scrollbars=no,resizable=no,width=300,height=200');

</SCRIPT>
</BODY>
</HTML>
```



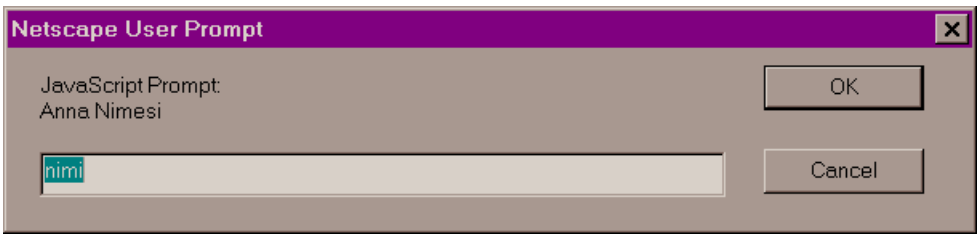
Edellisen esimerkin tulostama ikkuna, kun on vastattu ”OK” confirm-metodin kysymykseen.

---

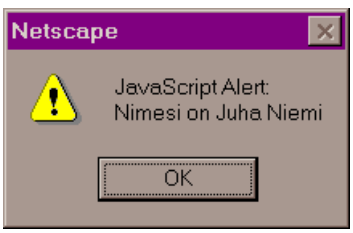
prompt(viesti, kentän oletussisältö**)**	Näyttää viesti-ikkunan, jossa on myös tekstikenttä, jonka avulla voidaan lukea käyttäjän syöttämää dataa.

Esimerkki.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">

var arvo = prompt("Anna Nimesi","nimi");

var tulos = "Nimesi on " + arvo;

alert(tulos);

</SCRIPT>
</BODY>
</HTML>
```



Esimerkin tuloste sivulle tultaessa.



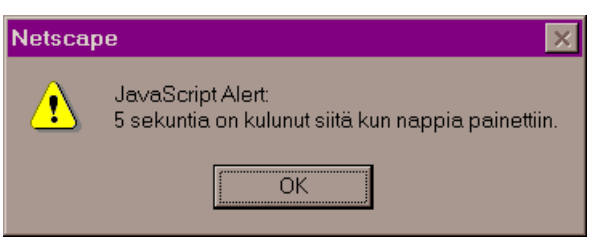
Esimerkin tuloste kun alkuarvon ”nimi” on korvattu tekstillä ”Juha Niemi” ja painettu OK-nappia.

---

tunnus=**setTimeout(**lause, aika**)**	
	Lause (annetaan merkkijonona) suoritetaan kun **millisekunteina** annettu aika on kulunut. **Tunnusta** käytetään peruuttamaan toiminto **clearTimeout()** metodin avulla.
clearTimeout(tunnus**)**	Nollaa sen **setTimeout()**-metodin, jonka annettu **tunnus** vastaa sulkeissa olevaa.

Esimerkki: Viivenapin painalluksella saadaan aikaan muistutus 5 sekunnin kuluttua napin painalluksesta. Peruutusnapin painallus peruuttaa toiminnon.

```
    <HTML>
    <HEAD>
    <!--    -->
    <SCRIPT LANGUAGE="JavaScript">
function huomio()
{
   alert("5 sekuntia on kulunut siitä kun nappia pai-
nettiin.");
}
</SCRIPT>
<!--  -->
</HEAD>

<BODY>
<FORM>

<INPUT TYPE="button"  VALUE="muistutus"
NAME="viivenappi"
   onClick="ajastin=setTimeout('huomio()',5000)">

<INPUT TYPE="button"  VALUE="peruutus"
NAME="peruutusnappi"
   onClick="clearTimeout(ajastin)">

</FORM>
</BODY>
</HTML>
```

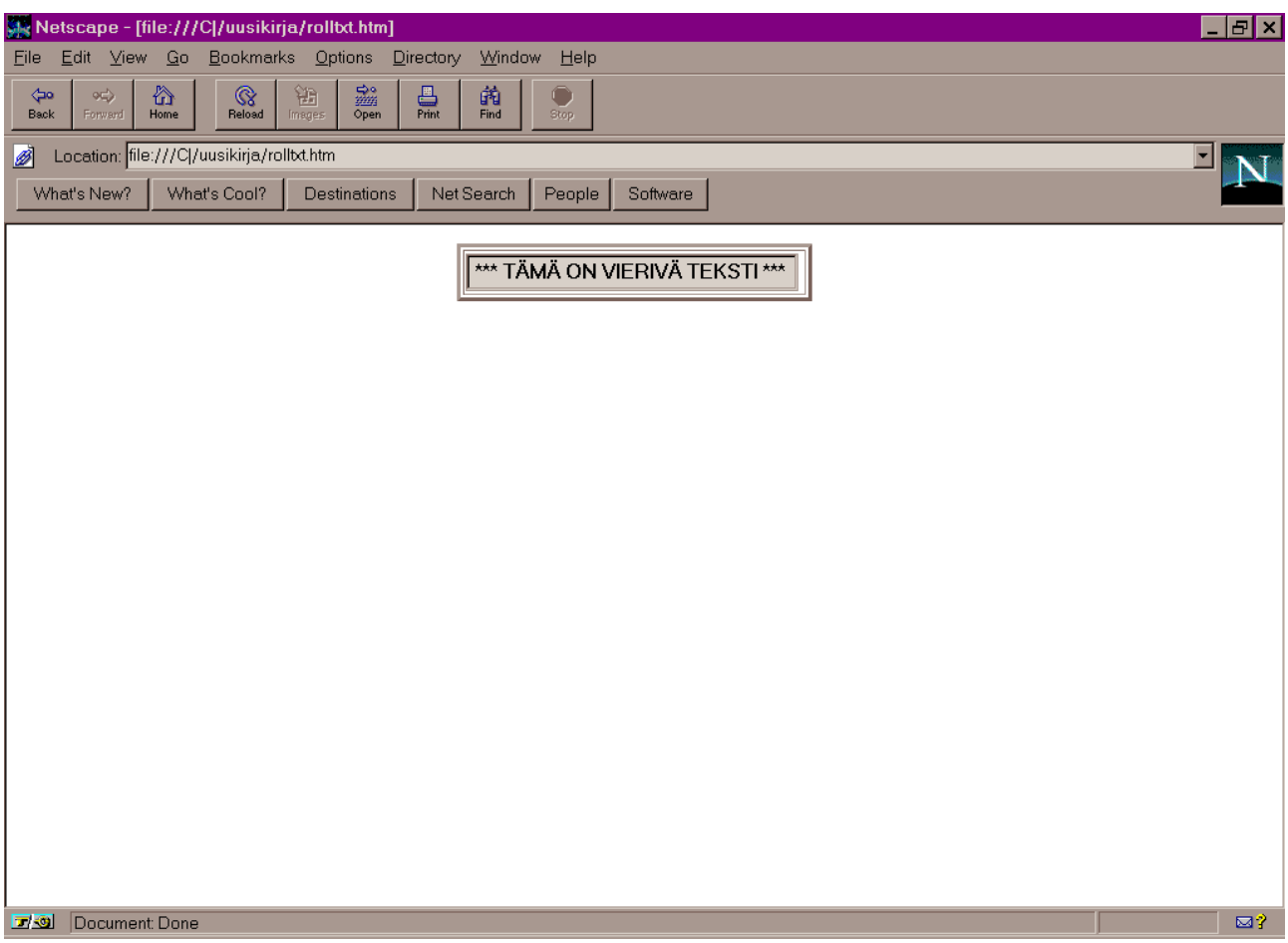


Edellisen esimerkin muistutus.

Esimerkki:Vierivän tekstin tulostus tekstikenttään.

```
<HTML>
<HEAD>
<!--   Juha Niemi  3.9.1996  -->
<SCRIPT LANGUAGE="javascript">
var apu=1;

function vaihda() {

var final_txt="";
var vali=250;
var txt="*** TÄMÄ ON VIERIVÄ TEKSTI *** ";


if(apu>0){

        for(var i=apu;i<txt.length;i++){
                final_txt+=txt.charAt(i);
                        }
        for(var j=0;j<apu;j++){
        final_txt+=txt.charAt(j);
        if(j==apu-1)
        final_txt+=" ";
                        }
                }

document.lomio.teksti.value=final_txt;

apu++;

if(apu==txt.length-1){
        apu=1;
document.lomio.teksti.value=" "+txt;
        vali=4000;
                }


viive=setTimeout("vaihda()",vali);

        }


</SCRIPT>
<!--    -->
```

```
</HEAD>
<BODY BGCOLOR="#ffffff">
<FORM NAME="lomio">
<CENTER><TABLE BORDER="3"><TR><TD>
<INPUT TYPE="text" SIZE="31" NAME="teksti" >
</TD></TR></TABLE></CENTER>
</FORM>
<SCRIPT LANGUAGE="javascript">

document.lomio.teksti.value=" *** TÄMÄ ON VIERIVÄ
TEKSTI *** ";

viive=setTimeout("vaihda()",2000);

</SCRIPT>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

Tapahtumafunktiot:

onLoad	Suorittaa määrätyn JavaScript-koodin, kun sivu tai kaikki sivut FRAMESET:ssä ladataan muistiin. Esimerkiksi <BODY onLoad="…">tai <FRAMESET onLoad="…">.
onUnload	Suorittaa määrätyn JavaScript-koodin, kun sivulta poistutaan. Käyttö kuten onLoad-tapahtumafunktion.

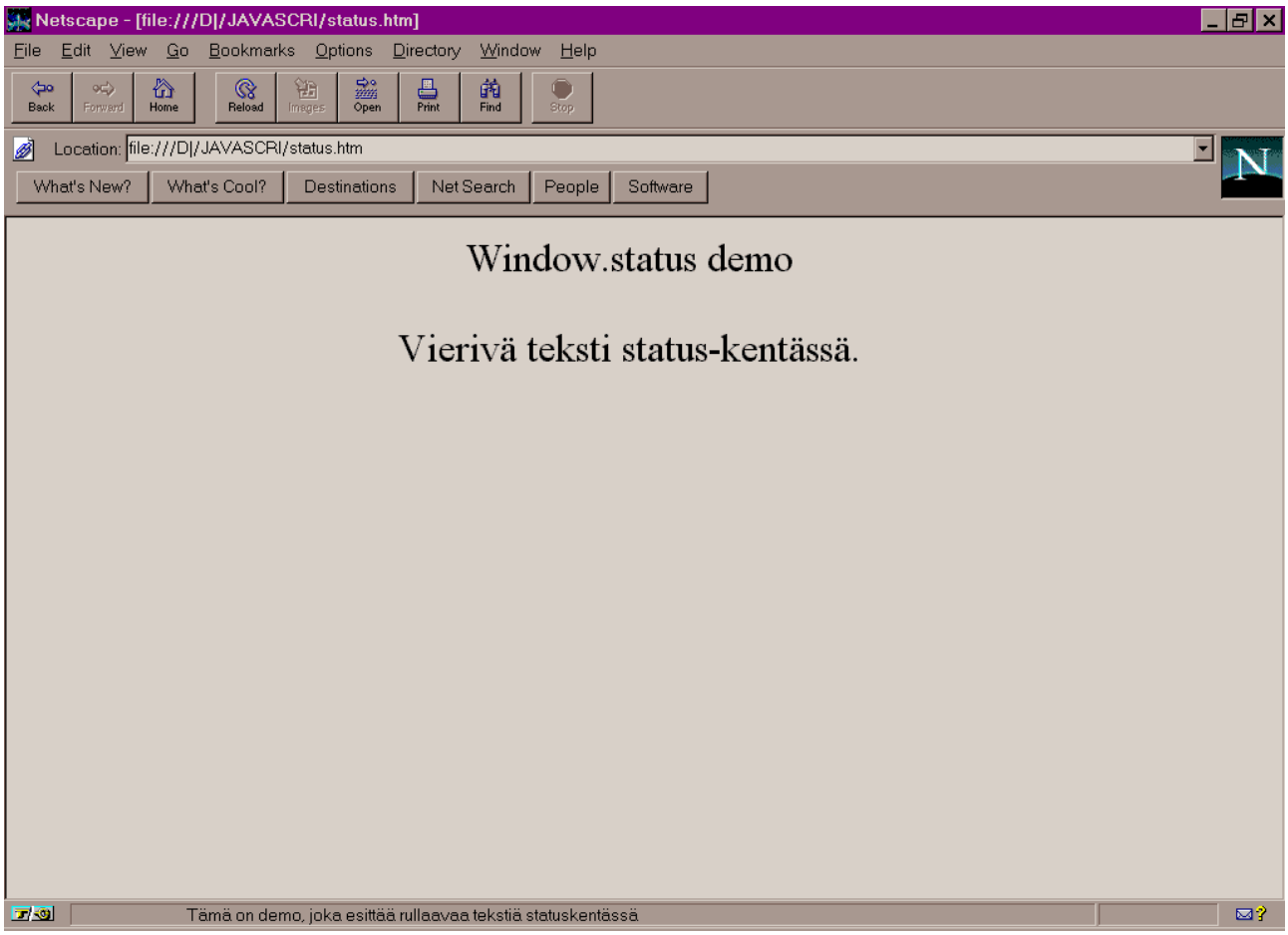
Esimerkki.

```
<BODY  onUnload="alert('Näkemiin')">
```

Esimerkki. Vierivän tekstin tulostus status-kenttään sivun latauksen yhteydessä.

```
<HTML>
<HEAD>
<!-- Juha Niemi 4.9.1996 -->
<SCRIPT LANGUAGE="JavaScript">


function rullaa(arvo)


{


        var teksti= "Tämä on demo, joka esittää rullaavaa
tekstiä statuskentässä";
        var tulos = " ";
        var i   = 1;

        if (arvo > 100) {
```

```
                arvo--;
                var cmd="rullaa(" + arvo + ")";
                ajastin_2=window.setTimeout(cmd,100);
        }
        else if (arvo <= 100 && arvo > 0) {
                for (i=0 ; i < arvo ; i++) {
                        tulos+=" ";
                }
                tulos+=teksti;
                arvo--;
                var cmd="rullaa(" + arvo + ")";
                    window.status=tulos;
                ajastin_2=window.setTimeout(cmd,100);
        }
        else if (arvo <= 0) {
                if (-arvo < teksti.length) {
                        tulos+=teksti.substring(-
arvo,teksti.length);
                        arvo--;
                        var cmd="rullaa(" + arvo + ")";
                        window.status=tulos;
                        ajas-
tin_2=window.setTimeout(cmd,100);
                }
                else {
                        window.status=" ";
                        ajas-
tin_2=window.setTimeout("rullaa(100)",75);
                }
        }
}


</SCRIPT>
<!--  -->
</HEAD>
<BODY on-
Load="ajastin=window.setTimeout('rullaa(100)',500);">
<CENTER><FONT size="6">Window.status demo <BR>
<BR>
Vierivä teksti status-kentässä.

</BODY>
</HTML>
```



Edellisen esimerkin tuloste.

Esimerkki. Värikoodien testaussivu. Uusi ikkuna avataan tulostukseen kun painonappia on painettu. Lisäksi uuteen ikkunaan on sijoitettu *SULJE*-nappi.

```
<HTML>
<HEAD>

<TITLE>Värikoodit</TITLE>
<SCRIPT LANGUAGE="JavaScript">

function vaihda_vari(vari){

var teksti_vari = "#000000";
var F_vari = "'"+vari+"'";
```

```
//if-lauseella vaihdetaan tummissa väreissä teksti val-
koiseksi

if(F_vari=="'#000000'" || F_vari=="'#800000'" ||
F_vari=="'#B22222'" || F_vari=="'#006400'" ||
F_vari=="'#004080'")teksti_vari="#ffffff";

tulos=window.open('','vari_ik','toolbar=no,location=no,di
rectories=no,menubar=no,status=no,copyhistory=no,scrollba
rs=no,resizable=no,width=100,height=150');


var runko="<BODY bgcolor="+F_vari+">";

tulos.document.write(runko+"<CENTER><B>","Väri".fontcolor
(teksti_vari),"<BR>",F_vari.fontcolor(teksti_vari),"</B><
BR>");

tulos.document.write("<FORM><TABLE border='4'><TR><TD>");

tulos.document.write("<TABLE border='2'><TR><TD>");

tulos.document.write("<INPUT type='button' value='SULJE'
onClick='window.close();'>");

tulos.document.write("</TD></TR></TABLE></TD></TR></TABLE
>");

tulos.document.write("</FORM></CENTER></BODY>");

}

</SCRIPT>
</HEAD>



<BODY bgcolor="#ffffff">

<CENTER>
<H2><font color="#ff0000">VÄRIKOODIEN TESTAUS</H2>
<FORM>
<TABLE border="3"><TR><TD><font color="#ff0000"><B>
COLOR</TD><TD><font color="#ff0000"><B>RGB-koodi</TD><TD>
<font color="#ff0000"><B>Tulosta</TR>
```

```
<TR><TD>
Cyan
</TD><TD>"#00FFFF"</TD><TD>
<INPUT type="hidden" name="vari_1" value="#00FFFF">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_1.value);">
</TD></TR>


<TR><TD>
Black
</TD><TD>"#000000"</TD><TD>
<INPUT type="hidden" name="vari_2" value="#000000">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_2.value);">
</TD></TR>


<TR><TD>
Blue
</TD><TD>"#0000FF"</TD><TD>
<INPUT type="hidden" name="vari_3" value="#0000FF">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_3.value);">
</TD></TR>


<TR><TD>
Chartreuse
</TD><TD>"#7FFF00"</TD><TD>
<INPUT type="hidden" name="vari_4" value="#7FFF00">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_4.value);">
</TD></TR>


<TR><TD>
Cornflowerblue
</TD><TD>"#6495ED"</TD><TD>
<INPUT type="hidden" name="vari_5" value="#6495ED">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_5.value);">
</TD></TR>
```

```
<TR><TD>
Lightblue
</TD><TD>"#ADD8E6"</TD><TD>
<INPUT type="hidden" name="vari_6" value="#ADD8E6">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_6.value);">
</TD></TR>


<TR><TD>
Lightyellow
</TD><TD>"#FFFFE0"</TD><TD>
<INPUT type="hidden" name="vari_7" value="#FFFFE0">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_7.value);">
</TD></TR>


<TR><TD>
Magenta
</TD><TD>"#FF00FF"</TD><TD>
<INPUT type="hidden" name="vari_8" value="#FF00FF">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_8.value);">
</TD></TR>


<TR><TD>
Maroon
</TD><TD>"#800000"</TD><TD>
<INPUT type="hidden" name="vari_9" value="#800000">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_9.value);">
</TD></TR>


<TR><TD>
Mediumpurple
</TD><TD>"#9370DB"</TD><TD>
<INPUT type="hidden" name="vari_10" value="#9370DB">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_10.value);">
</TD></TR>



<TR><TD>
```

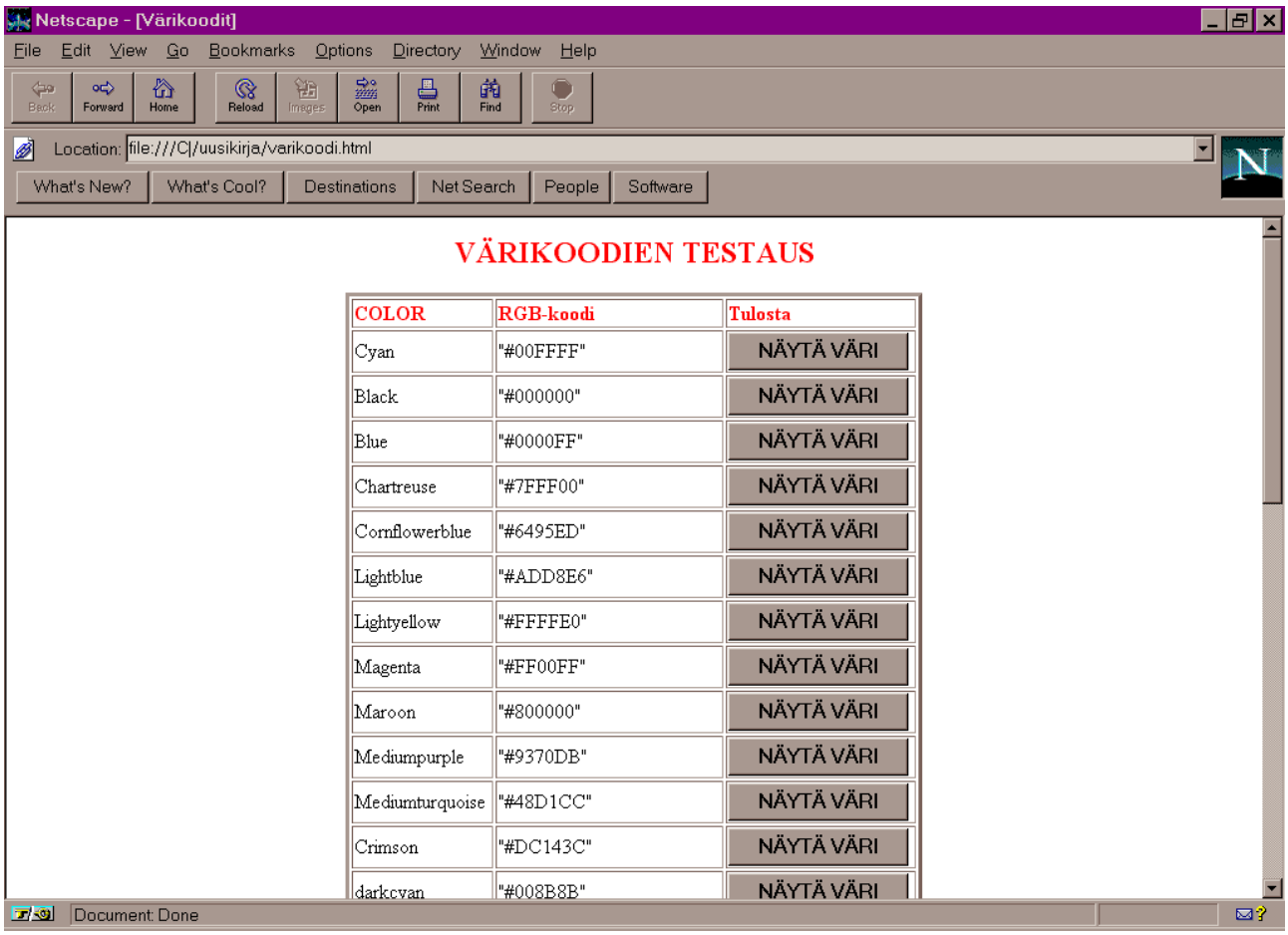
```
Mediumturquoise
</TD><TD>"#48D1CC"</TD><TD>
<INPUT type="hidden" name="vari_11" value="#48D1CC">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_11.value);">
</TD></TR>


<TR><TD>
Crimson
</TD><TD>"#DC143C"</TD><TD>
<INPUT type="hidden" name="vari_12" value="#DC143C">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_12.value);">
</TD></TR>


<TR><TD>
darkcyan
</TD><TD>"#008B8B"</TD><TD>
<INPUT type="hidden" name="vari_13" value="#008B8B">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_13.value);">
</TD></TR>


<TR><TD>
Darkgray
</TD><TD>"#A9A9A9"</TD><TD>
<INPUT type="hidden" name="vari_14" value="#A9A9A9">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_14.value);">
</TD></TR>


<TR><TD>
Darkgreen
</TD><TD>"#006400"</TD><TD>
<INPUT type="hidden" name="vari_15" value="#006400">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_15.value);">
</TD></TR>



<TR><TD>
Darkorange
```

```
</TD><TD>"#C95C14"</TD><TD>
<INPUT type="hidden" name="vari_16" value="#C95C14">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_16.value);">
</TD></TR>


<TR><TD>
Darkpink
</TD><TD>"#FF1493"</TD><TD>
<INPUT type="hidden" name="vari_17" value="#FF1493">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_17.value);">
</TD></TR>


<TR><TD>
Firebrick
</TD><TD>"#B22222"</TD><TD>
<INPUT type="hidden" name="vari_18" value="#B22222">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_18.value);">
</TD></TR>


<TR><TD>
Gold
</TD><TD>"#FFD700"</TD><TD>
<INPUT type="hidden" name="vari_19" value="#FFD700">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_19.value);">
</TD></TR>


<TR><TD>
Greenyellow
</TD><TD>"#ADFF2F"</TD><TD>
<INPUT type="hidden" name="vari_20" value="#ADFF2F">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_20.value);">
</TD></TR>



<TR><TD>
Hotpink
</TD><TD>"#FF69B4"</TD><TD>
```

```
<INPUT type="hidden" name="vari_21" value="#FF69B4">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_21.value);">
</TD></TR>


<TR><TD>
Navy
</TD><TD>"#004080"</TD><TD>
<INPUT type="hidden" name="vari_22" value="#004080">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_22.value);">
</TD></TR>


<TR><TD>
Orange
</TD><TD>"#FFA500"</TD><TD>
<INPUT type="hidden" name="vari_23" value="#FFA500">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_23.value);">
</TD></TR>


<TR><TD>
Pink
</TD><TD>"#FFC0CB"</TD><TD>
<INPUT type="hidden" name="vari_24" value="#FFC0CB">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_24.value);">
</TD></TR>


<TR><TD>
Red
</TD><TD>"#FF0000"</TD><TD>
<INPUT type="hidden" name="vari_25" value="#FF0000">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_25.value);">
</TD></TR>



<TR><TD>
Rosybrown
</TD><TD>"#BC8F8F"</TD><TD>
<INPUT type="hidden" name="vari_26" value="#BC8F8F">
```

```
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_26.value);">
</TD></TR>


<TR><TD>
Salmon
</TD><TD>"#FA8072"</TD><TD>
<INPUT type="hidden" name="vari_27" value="#FA8072">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_27.value);">
</TD></TR>


<TR><TD>
Silver
</TD><TD>"#C0C0C0"</TD><TD>
<INPUT type="hidden" name="vari_28" value="#C0C0C0">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_28.value);">
</TD></TR>


<TR><TD>
Tan
</TD><TD>"#D2B48C"</TD><TD>
<INPUT type="hidden" name="vari_29" value="#D2B48C">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_29.value);">
</TD></TR>


<TR><TD>
Tomato
</TD><TD>"#FF6347"</TD><TD>
<INPUT type="hidden" name="vari_30" value="#FF6347">
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_30.value);">
</TD></TR>



<TR><TD>
Yellow
</TD><TD>"#FFFF00"</TD><TD>
<INPUT type="hidden" name="vari_31" value="#FFFF00">
```

```
<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_31.value);">
</TD></TR>


<TR><TD>
Oma valinta
</TD><TD><INPUT type="text" name="vari_32" va-
lue="#FFFFFF"></TD><TD>

<INPUT type="button" value="NÄYTÄ VÄRI" on-
Click="vaihda_vari(this.form.vari_32.value);">
</TD></TR>


<!-- Lopuksi omavalintaisen värikoodin (tai tekstin)
syöttö text-olion avulla -->


</TABLE></FORM></B>
Kirjoita kenttään haluamasi värikoodi (heksana #FFFFFF)
 tai teksti (esim. white)

</CENTER>
</BODY>
</HTML>
```



Edellisen esimerkin tuloste kun mitään nappia ei ole painettu.

Edellisen esimerkin tulostusikkuna (joka tulee vanhan päälle) kun painettiin NÄYTÄ VÄRI -nappia Lightblue riviltä.

4.13 Cookie-mekanismi

WWW:n ongelma oli pitkään, että kun palvelin oli lähettänyt lukuohjelmalle dokumentin, se unohti välittömästi tapahtuman. Tarvittiin mekanismi, joka varastoi tietoa yhteyksistä palvelimen ja lukuohjelmien välillä, jotta voitaisiin toteuttaa monimutkaisiakin interaktiivisia dokumentteja.

Tämä ongelma ratkaistiin Cookie-mekanismilla. Nimi Cookie ei ole lyhenne mistään vaan on eräs nimi, joka on otettu syystä tai toisesta käyttöön tässä yhteydessä.

Cookie-mekanismissa lukuohjelma tallentaa sivukohtaista informaatiota jokaisesta sivusta, jolla on vierailtu ja jos näille sivuille mennään uudestaan lähetetään palvelimelle myös tätä varastoitua informaatiota kun pyydetään asianomaista dokumenttia.

Liikenne palvelimen ja lukuohjelman välillä tapahtuu HTML-protokollalla. Kun lukuohjelma lähettää pyynnön tietystä dokumentista palvelimelle se liittää pyyntöön niin sanottuja otsikkokenttiä, jotka sisältävät pyyntöä koskevaa informaatiota. Palvelin hakee pyydetyn dokumentin ja lähettää sen lukuohjelmalle liittäen myös otsikkokenttiä dokumentin edellä lähetettäväksi. Otsikkokentät sisältävät pyydettyyn dokumenttiin liittyvää informaatiota, kuten MIME-tyypin. Otsikkokentissä on määritelty nimikenttä ja varsinaiset informaatiokentät. Otsikkokentät ovat yleisesti seuraavaa muotoa.

```
Kentän_Nimi: info_1; info_2; … info_n
```

Myös Cookie-informaatio välitetään otsikkokentissä. Kun lukuohjelma pyytää sivua ensimmäistä kertaa lähettää palvelin Set-Cookie otsikkokentän (yhden tai useampia), joka sisältää eri informaatiokenttiä kuten ”parasta ennen” päiväys, hakemistopolku ja palvelintietoa. Lukuohjelma tallentaa nämä tiedot dokumenttikohtaisesti paikallisesti käyttäjän levylle. Kun lukuohjelma pyytää palvelimelta dokumentteja se liittää pyyntöön Cookie-otsikkokentän, jos pyydetyllä dokumentilla on varastoituna sellainen. (eli jos sivulla on käyty aikaisemmin).

Palvelimen lähettämä Set-Cookie -otsikkokenttä sisältää seuraavat informaatiokentät. Yhteen palvelupyyntöön voidaan liittää useita Set_Cookie -otsikkokenttiä.

Set-Cookie: name=arvo; [expires=arvo; path=arvo; domain=arvo;
secure]

Informaatiokentistä ainoastaan **name**-kenttä on aina mukana. Se sisältää merkkijonon, jota käytetään tunnistukseen. [puolipiste (;), pilkku (,) ja välilyönti eivät ole sallittuja merkkejä].

Expires-kenttä sisältää ”parasta ennen” päiväyksen. Kun tämä päivä on ohitettu ei tätä dokumenttia koskevaa Cookie-informaatiota enää säilytetä eikä lähetetä palvelimelle . Päiväys on muotoa (ainoastaan GMT-aika) . Viikon päivät ja kuukaudet englanniksi lyhentein. Sun, Mon, Tue, Wed, Thu, Fri ja Sat sekä Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov ja Dec.

Wdy, DD-Mon-YY HH:MM:SS GMT

Oletusarvoisesti ”parasta ennen” päiväys asetetaan kunkin surffailusession loppuun.

Path-kentässä on sen URL:n polkuosa, johon Cookie liittyy.

Domain-kentässä on sen URL;n domainosa, johon Cookie liittyy. Oletusarvo on sen dokumentin domainosa, joka varastoi Cookien.

Secure-kenttä määrittelee, että Cookie voidaan lähettää vain palvelimille, jotka käyttävät SSL-protokollaa (HTTPS-palvelimet).

Lukuohjelman lähettämä Cookie-otsikkokenttä on seuraava.

```
Cookie: name_1=arvo; name_2=arvo; … name_n=arvo
```

Cookie, jolla on sama polku ja nimi korvaa aikaisemman.

Netscape Navigator 3.0 varastoi 300 Cookie-informaatiota maksimissaan. Jokaisen Cookien koko on rajoitettu 4 kilotavuun. Domain:ia kohti varastoidaan korkeintaan 20 Cookie:ta. Kun määrät ylittyvät lukuohjelma pyyhkii pois Cookie:ja vanhimmasta aloittaen. Jos Cookie:n koko ylittää maksimin, se sovitetaan maksimiin.

Cookie-mekanismia käytetään sovelluksissa pääasiassa muistina eli sen avulla voidaan varastoida informaatiota myöhempää tarkastelua silmällä pitäen.

4.13.1 Cookie:n toimeenpano CGI-ohjelmalla

Jotta Cookie-mekanismia voitaisiin hyödyntää pitää palvelimen pystyä hyödyntämään saamansa Cookie-otsikkokentät informaationeen ja generoimaan tarvittavat Set-Cookie -otsikkokentät. Tämä toteutetaan pääsääntöisesti CGI-ohjelmien avulla.

Jos esimerkiksi perustetaan WWW:tä tutkiva hakupalvelu, joka hakee käyttäjän määrittelemiä dokumentteja. Voidaan pystyttää systeemi, joka noudattaa seuraavaa kaavaa.

1. Käyttäjä saapuu palveluun URL:n kautta, joka käynnistää CGI-ohjelman.

2. CGI-Ohjelma tarkistaa onko käyttäjä käyttänyt palvelua aikaisemmin tarkistamalla onko otsikkokentissä Cookie-kenttää.

3. Jos Cookie-kenttää ei ole, lähetetään käyttäjälle aloitussivu.
4. Jos otsikkokentissä on Cookie-kenttä, Cookie luetaan ja

käyttäjälle lähetetään sivu, joka on siinä tilassa johon se jäi kun käyttäjä vieraili palvelussa edellisen kerran.

5. Kun käyttäjä suorittaa hakutoiminnon, lähetetään vastaussivun yhteydessä Set-Cookie -otsikkokenttä, jolla korvataan vanha Cookie ja näin tiedot tulevat päivitetyiksi.

4.13.2 Cookie:n toimeenpano JavaScriptillä

JavaScriptiä käyttämällä voidaan Cookie-mekanismi toteuttaa lukuohjelmassa ilman CGI-ohjelmia. Tämä hoidetaan **document**-olion **cookie**-muuttujan avulla. Cookie-muuttujaan sijoitetaan merkkijono, joka vastaa Cookie-otsikkokentän arvoa koskien kyseessä olevaa sivua. Arvoksi annettavan merkkijonon pitää vastata sitä, jonka palvelin lähettäisi vastaavassa tapauksessa Set-Cookie -otsikkokentässä.

Esimerkki.

```
document.cookie='cookie1=Eka_cookie; expires=Mon, 01-Nov-96 12:00:00 GMT; path="/" ';
```

Näin muodostetaan Cookie, jonka nimi on cookie1 ja jonka "parasta ennen" päiväys on 1.11.1996 klo 12:00:00 GMT ja se on voimassa kaikilla domainin dokumenteilla, koska poluksi on annettu juuri (/).

"Parasta ennen" päiväys voidaan antaa hyödyntämällä **Date**-oliota varsinkin jos päivämäärä on kaukana tulevaisuudessa.

Esimerkki.

```
expires= new Date();
expires.setTime(expires.getTime() +24 * 60 * 60 * 365 * 1000);
document.cookie=”cookie2=Toka_cookie; expires=” + expires.toGMTString();
```

Luku 24*60*60*365*1000 on vuosi millisekunteina eli ”parasta ennen” päivays asetetaan tästä hetkestä vuodella eteen päin. Metodi toGMTString() hoitaa merkkijonon oikeaan GMT-muotoon.

Cookie-mekanismia hyödynnettäessä tarvitaan usein samoja toimintoja suorittavia funktioita kuten luo uusi Cookie ja lue olemassa oleva Cookie. Näitä funktioita loytyy JavaScriptille kirjoitettuna verkosta esimerkiksi osoitteesta (**http://www.hidaho.com/cookies/cookie.txt**).

5. CGI-ohjelmat

5.1 Yleistä

CGI (*Common Gateway Interface*) on standardi, joka määrittelee miten palvelimen ja ohjelman välinen kommunikointi tapahtuu. Tästä johtuen WWW-palvelimessa sijaitsevia ohjelmia kutsutaan usein CGI-ohjelmiksi.

WWW-palvelimessa määritellään hakemisto, yleensä **cgi-bin**, josta haettavat tiedostot ajetaan ohjelmina palvelinkoneessa ja niiden tuottamat tulosteet lähetetään lukuohjelmalle. Normaalistihan palvelin lähettää pyydetyn tiedoston lukuohjelmalle sellaisenaan. Palvelimessa voidaan myös määrätä tietty tarkennin, yleensä ***.cgi**, jolla varustetut tiedostot ajetaan ohjelmina. Itse CGI-ohjelmat voidaan kirjoittaa millä tahansa ohjelmointikielellä, kuten C ja C++ tai scripteinä, kuten unix-komentotiedostot tai perl-kielellä (scriptaus kieli, mutta vaatii oman perl-tulkin). Tässä kirjassa käytetään unix-komentotiedostoja scripteinä ja C- sekä C++ -kielillä tehtyjä ohjelmia.

Ohjelmahakemiston nimeksi voidaan antaa jokin muukin kuin cgi-bin. Tämä seikka kannattaa tarkistaa palvelimen ylläpitäjältä kuten sinne kirjoittamiseen tarvittavat oikeudetkin. Ohjelmahakemistoon kirjoittamiseen tarvittavat oikeudet pitää yleensä pyytää erikseen palvelimen ylläpitäjältä. Tässä kirjassa oletetaan (tämä on yleensä myös oletusasetus palvelimen määrittelytiedostoissa), että ohjelmahakemiston nimi on cgi-bin. Tämän kirjan esimerkkien testauksessa on käytössä ollut **ncsa**-palvelin ja **Solaris 2.4** ja **Solaris 2.5** käyttöjärjestelmät. Seuraavassa Esimerkki URL, joka johtaa palvelimen ohjelmahakemistossa olevan ohjelman ajamiseen palvelinkoneessa ja sen tulosteiden lähettämiseen lukuohjelmalle.

http://www.kone.fi/cgi-bin/heippa

Jos palvelimeen on määritelty tietty tarkennin, esimerkiksi ***.cgi**, voidaan tällä tarkentimella varustetut tiedostot ajaa ohjelmina muistakin kuin ohjelmahakemistosta. Esimerkiksi ohjelma heippa.cgi voitaisiin sijoittaa käyttäjän kotihakemistoon. Tällöin se on sijoitettava kotihake-

mistossa alihakemistoon (kuten kaikki muutkin www-palvelimen noutamat tiedostot), joka on palvelimen määrittelytiedostoissa määrätty käyttäjien kotihakemistoksi (oletusarvo **ncsa**-palvelimessa **public_html**, mutta usein käytetään vain nimeä **html**). Tähän tiedostoon johtava URL olisi silloin seuraava.

http://www.kone.fi/~kayttajatunnus/heippa.cgi

Kaikille CGI-ohjelmille yhteinen osa on **Content-type** -otsikkokentän tulostus ensimmäiseksi. Siinä kerrotaan MIME-tyyppi ja tämän jälkeen pitää seurata tyhjä rivi.

MIME-tyyppi kertoo lukuohjelmalle minkälaista tietoa on tulossa. Jos tulossa on pelkkää muotoilematonta tekstiä, voidaan käyttää MIME-tyyppiä **text/plain**. Jos taas tulossa on HTML-koodia, käytetään MIME-tyyppiä **text/html**. Viimeksi mainittua tyyppiä käytetään vaikka koodi sisältäisikin **gif**- tai **jpg**-kuvia.

Pakollisten ensimmäisten rivien tulostus voidaan hoitaa seuraavasti unix-komentotiedostoissa.

```
#!/bin/sh
echo ”Content-type: text/html”
echo ””
```

Tai C-kielessä seuraavasti:

```
printf(”Content-type: text/html\n”);
printf(”\n”);
```

Tai C++ -kielessä seuraavasti:

```
cout<<” Content-type: text/html\n”;
cout<<”\n”;
```

Näiden rivien jälkeiset tulosteet ohjataan suoraan lukuohjelmalle.

5.2 CGI-ohjelma unix-komentotiedostoilla

Unix-komentotiedosto on yksinkertaisimmillaan lista komentoja, jotka suoritetaan peräkkäin. Komentotiedostoja voidaan kirjoittaa eri komentotulkeille, jotka käyttävät kukin omia komentotiedostojaan. Tässä kirjassa keskitytään vain **sh** (*Bourne shell*) -komentotulkkiin, joka on unixin peruskomentotulkki ja löytyy kaikista unix-koneista. Sh-tulkin johdannaiset, kuten **ksh** (*Korn shell*) ja **bash** (*Bourne again shell*) ymmärtävät sh-komentotulkille tehdyt komentotiedostot sellaisinaan.

Komentotiedosto voi sisältää joko tulkin sisäisiä komentoja tai ulkoisia komentoja. Ulkoiset komennot ovat muita ajettavia ohjelmia. Komennot erotetaan toisistaan kirjoittamalla ne eri riveille. Komentotiedostoja voidaan kirjoittaa millä tahansa teksti-editorilla, joka tallentaa tekstin muotoilemattomana (esimerkiksi Windowsin **Notepad** tai unixin peruseditori **vi**). Notepadia tai muuta DOS/Windows-ympäristön editoria käytettäessä pitää huomioida tämän ympäristön eri rivinvaihtomerkit, jotka näkyvät unix-koneessa **^M** (Ctrl-M) - merkkeinä. Nämä pitää yleensä poistaa ainakin komentotiedoston kolmelta ensimmäiseltä riviltä. ^M -merkit saa näkyviin vi:llä.

Kun komentotiedosto on kirjoitettu ja tallennettu (ja siirretty unix-koneelle), sille pitää antaa suoritusoikeus ennen kuin se voidaan ajaa. Tämä voidaan toteuttaa komennolla,

```
chmod 755 tiedoston_nimi
```

joka antaa tiedoston omistajalle kaikki oikeudet tiedostoon ja ryhmälle, johon omistaja kuuluu, sekä kaikille muille luku ja suoritusoikeuden tiedostoon. Antamalla tämä komento kaikille CGI-ohjelmana ajettaville

tiedostoille vältetään yleensä oikeuksien puuttumisesta johtuvat virheilmoitukset. Tiedostolle voidaan antaa myös komento,

```
chmod 711 tiedoston_nimi
```

joka antaa ryhmälle ja muille vain suoritusoikeuden ilman lukuoikeutta ja tiedoston omistajalle edelleen kaikki oikeudet. Lukuoikeuksien puuttuminen saattaa joskus aiheuttaa virheilmoituksia. Tiedostojen voimassa olevat oikeudet saadaan näkyviin hakemistolistauskomennolla ls ja lisäämällä siihen parametrit **la**
seuraavasti.

```
ls -la
```

Komento tuottaa esimerkiksi seuraavan tulosteen.

```
total 210
drwxr-xr-x      2 jniemi   user      25 Sep 10 20 1132 temp
-rwxr-xr-x      1 jniemi   user      87 Sep 10 20 13:26 koe
-rwx--x--x      1 jniemi   user      53 Sep 10 20 14:37 koescript
```

Tulosteessa näkyy tiedoston oikeudet (-rwxr-xr-x) ja omistaja (jniemi), ryhmä (user), tiedoston koko, viimeinen käsittelypäivä kellonaikoineen ja tiedoston nimi. Oikeuksien kohdalla ensimmäinen kirjain d merkitsee hakemistoa ja viiva ensimmäisessä sarakkeessa tiedostoa. Loppuosa oikeuksista on jaettu kolmen merkin ryhmiin rwx|rwx|rwx, joista ensimmäinen ryhmä rwx edustaa tiedoston omistajaa ja toinen rwx ryhmää ja kolmas rwx kaikkia muita. Kukin oikeus on voimassa jos kyseinen kirjain on merkitty ja oikeus on poistettu jos kirjaimen paikalla on viiva (-). Oikeuksien tunnukset ovat seuraavat.

r	Lukuoikeus
w	Kirjoitusoikeus
x	Suoritusoikeus

Kirjaimia edustavat myös seuraavat numerot.

4	Lukuoikeus
2	Kirjoitusoikeus
1	Suoritusoikeus

Näin ollen voidaan antaa komento chmod 755 tiedosto_nimi, jossa kolme numeroa edustaa omistajaa, ryhmää ja muita. Asianomaisella paikalla oleva luku muodostuu oikeuksien summasta. Edellisellä komennolla saa omistaja kaikki oikeudet, koska 4+2+1=7, ja ryhmä ja muut vain luku- ja suoritusoikeudet, koska 4+1=5.

Listaus komennon voi antaa myös yksittäiselle tiedostolle tai tiedosto ryhmälle käyttäen korvausmerkkiä (*)

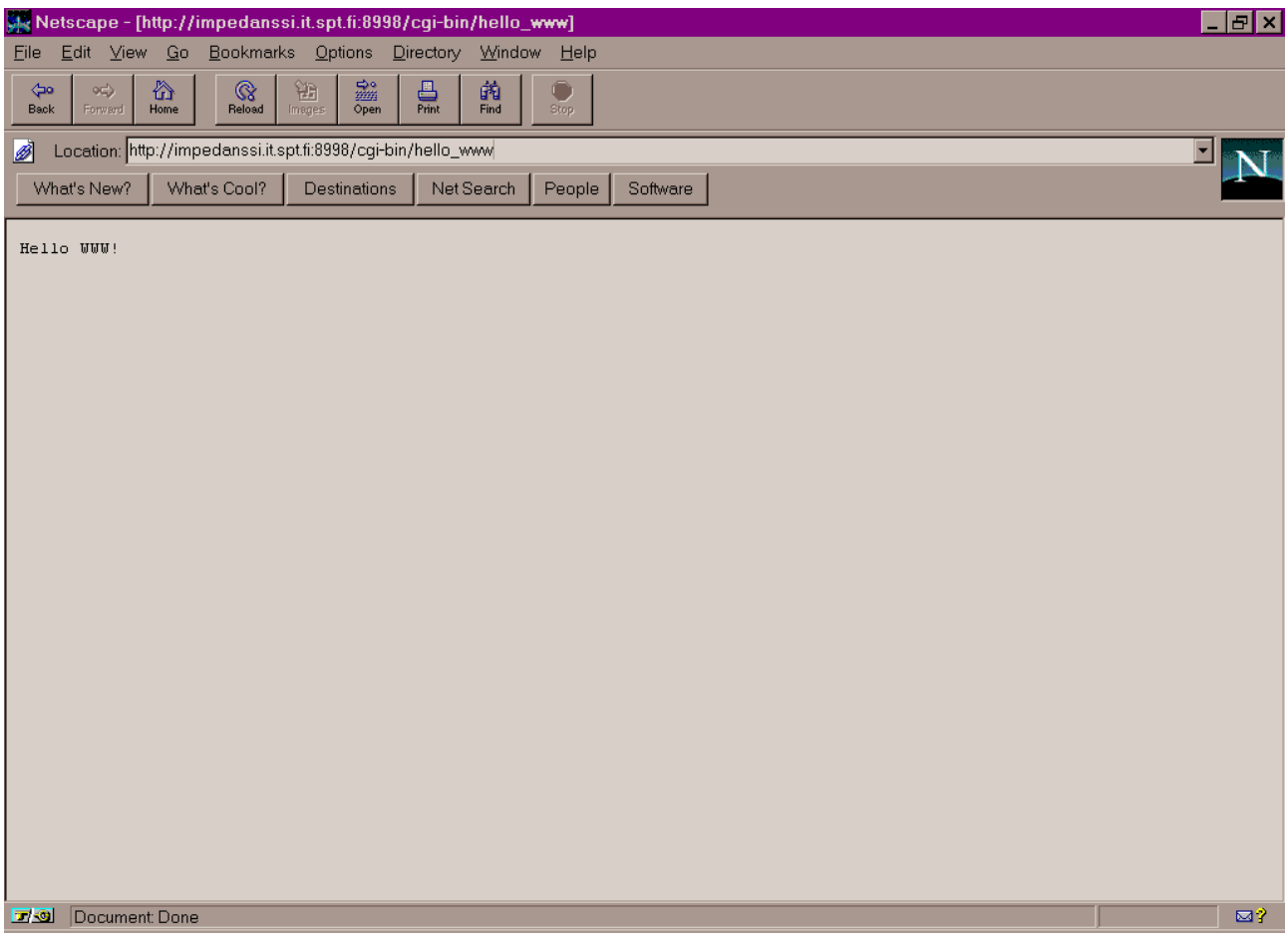
```
ls -la tiedosto_nimi

ls -la tied*
```

5.2.1 Ensimmäinen komentotiedosto

Komentotiedosto aloitetaan **#!/bin/sh** -rivillä, joka kertoo suorittavan komentotulkin nimen. Seuraavaksi tulevat CGI-ohjelmien pakolliset rivit **Content-type: text/html** ja **tyhjä rivi**. Tulostuskäskynä käytetään **echo** -komentoa, jossa lainausmerkkien sisään sijoitetaan tulostettava teksti.

Esimerkki. Jos halutaan tulostaa www-sivulle teksti ”Hello WWW!”. Kirjoitetaan seuraava komentotiedosto ja tallennetaan se vaikka nimellä hello_www palvelimen ohjelmahakemistoon (cgi-bin). Tämän jälkeen annetaan sille tarvittavat oikeudet komennolla **chmod 755 hello_www**, jonka jälkeen CGI-ohjelma on valmis ajettavaksi kirjoittamalla lukuohjelmaan URL:ksi **http://www.kone.fi/cgi-bin/hello_www**.

```
#!/bin/sh
echo "Content-type: text/plain"
echo ""
echo "Hello WWW!"
```



Edellisen esimerkin tuloste.

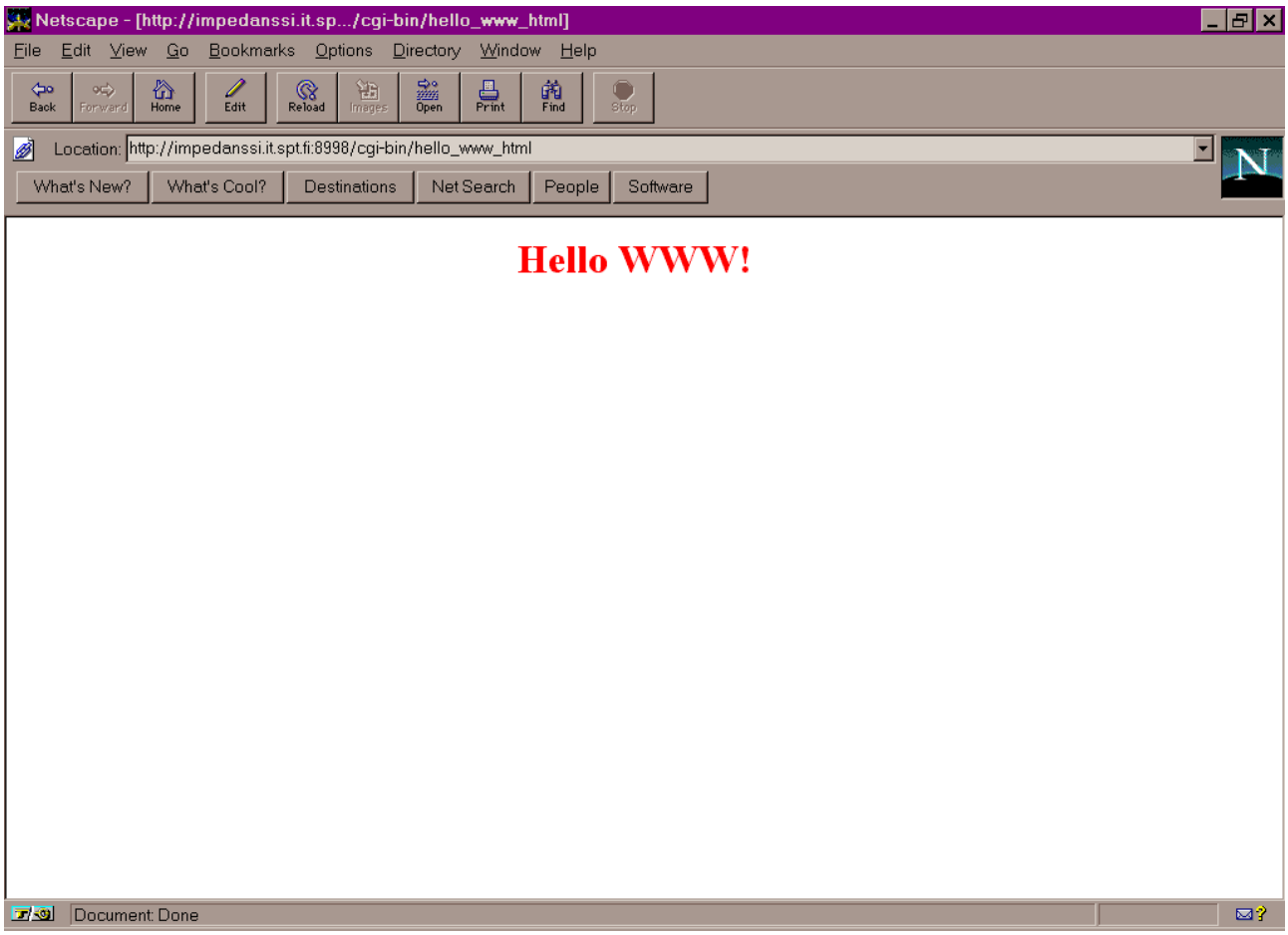
Tämä CGI-ohjelma tulostaa lukuohjelmaan pelkän tekstin "Hello_WWW!" ilman mitään muotoiluja. Komentotiedosto voidaan luonnollisesti ajaa myös suoraan komentoriviltä, jolloin usein vaaditaan ensimmäiseksi riviksi rivi, joka sisältää vain kaksoispisteen (:). Komentoriviltä ajettaessa kaikki lainausmerkkien sisällä oleva teksti tulostuu sellaisenaan. Rivi, joka sisältää kaksoispisteen aiheuttaa taas yleensä server error -

ilmoituksen www-sivulle, jos se on mukana CGI-ohjelmana ajettavassa komentotiedostossa.

```
:
#!/bin/sh
echo "Content-type: text/plain"
echo ""
jne.
```

Seuraavaksi tulostetaan sama teksti muotoiltuna käyttäen HTML-komentoja. Muokataan tiedosto seuraavanlaiseksi ja talletetaan se vanhan päälle.

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<HTML><BODY bgcolor='white'>"
echo "<P><CENTER><B>"
echo "<FONT color='#ff0000'>"
echo "<H1>Hello WWW!</H1>"
echo "</B></CENTER></BODY></HTML>"
```



Edellisen esimerkin tuloste.

Nyt saadaan tulostettavan www-sivun taustaväri valkoiseksi ja fontin väri punaiseksi sekä fontin koko hieman isommaksi. Lisäksi teksti keskitetään. Komennon lopetustagit eivät tässäkään ole aina välttämättömiä, mutta suotavia. HTML-tageissa pitää käyttää heitto (‘) -merkkejä, koska echo komennossa on lainausmerkit jo käytössä.

5.2.2 Käyttöjärjestelmäkomennot ja muuttujat komentotiedostossa

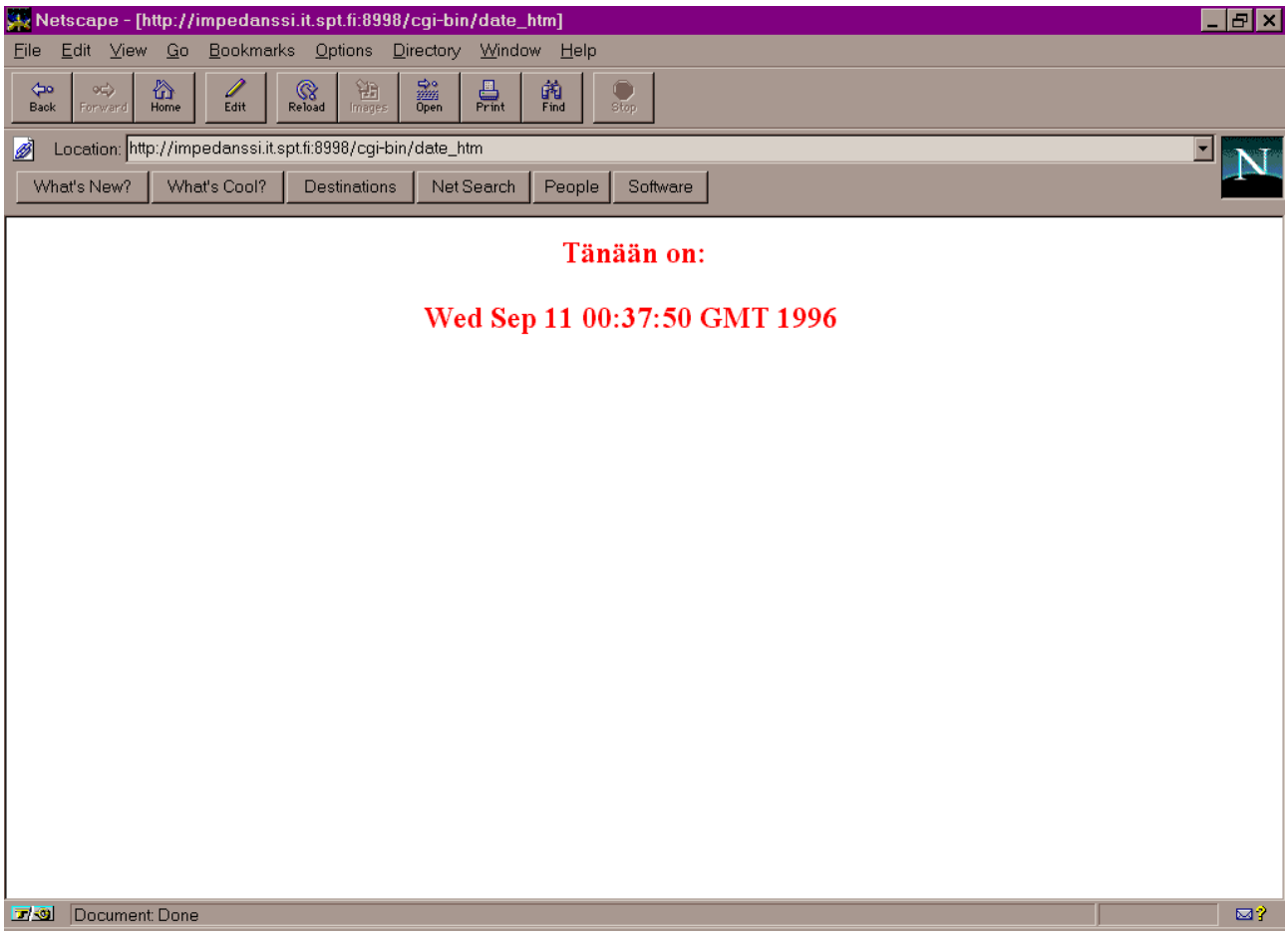
Käyttöjärjestelmän komennon tuloste voidaan myös sisällyttää CGI-ohjelmana ajettavaan komentotiedostoon. Esimerkiksi komento date, joka tulostaa päiväyksen muodossa 10 Sep 1996 20:53:00 GMT.

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<HTML><BODY bgcolor='#ffffff'>"
echo "<P><CENTER><B>"
echo "<FONT color='#ff0000' size='5'>"
echo "Tänään on<BR>"
date
echo "</B></CENTER></BODY></HTML>"
```

Muuttuja otetaan käyttöön komentotiedostossa yksinkertaisesti sijoittamalla siihen jokin arvo. Muuttujat ovat tyypittömiä eli muuttujaan voi tallettaa lukuarvoja tai merkkitietoa. Kun muuttujan arvoon viitataan, sijoitetaan sen nimen eteen dollari ($) -merkki. Myös ympäristömuuttujat, kuten PATH ja HOME ovat käytössä. Ympäristömuuttujien nimet ovat yleensä kirjoitettu isoilla kirjaimilla. Käyttöjärjestelmäkomennon tuloste voidaan myös sijoittaa muuttujan arvoksi sijoittamalla komento ` ` -merkkien sisään. (**HUOM! Ei heittomerkki vaan paina yhtäaikaa SHIFT ja merkki + -merkin oikealla puolella (*gravis*-merkki**). Gravis-merkkien välien osuus suoritetaan aina sellaisenaan.

Esimerkki.

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
paiva=`date`
echo "<HTML><BODY bgcolor='#ffffff'>"
echo "<P><CENTER><B>"
echo "<FONT color='#ff0000' size='5'>"
echo "Tänään on<BR>"
echo $paiva
echo "</B></CENTER></BODY></HTML>"
```



Edellisen esimerkin tuloste.

Pitkien tiedostojen kirjoittaminen on hankalaa **echo**-komentoa käyttämällä, mutta onneksi unix tarjoaa tulostuskäyttöön myös komennon **cat**. Normaalisti komento cat tulostaa vakiotulosteeseen (näyttö) sille argumenttina annetun tiedoston. Sillä voi myös ohjata tulosteen tiedostoon ja lisäksi voidaan käyttää syötteen ohjausta <<xx -mekanismilla.

Tiedoston tulostus.

```
cat tiedoston_nimi
```

Tulosteen ohjaus tiedostoon.

```
cat > tiedoston_nimi
```

Jos tiedostoa ei ole olemassa, komento ottaa vastaan merkkejä vakiosyötteestä (näppäimistö) niin kauan kunnes painetaan Ctrl D. Jos tiedosto on olemassa, uusi kirjoitetaan vanhan päälle.

Tulosteen ohjaus tiedostoon siten, että uusi tieto sijoitetaan vanhan perään.

```
cat >> tiedosto_nimi
```

Syötteen ohjaus <<xx -mekanismilla tarkoittaa sitä, että kahden (<) -merkin jälkeen annetaan tunniste ja kaikki tämän jälkeen tulevat rivit ohjataan tulosteeseen kunnes tunniste kohdataan uudelleen. Muuttujat voidaan edelleen tulostaa liitämällä $ -merkki niiden nimen eteen.

Esimerkki. Tulostetaan teksti tunnisteiden (EndOfHeader) välissä.

```
cat << EndOfHeader
Kaikki tämä
tulostuu nyt
suoraan kunnes tunniste
kohdataan
EndOfHeader
```

Esimerkki. Edellinen Esimerkki toisin kirjoitettuna.

```
#!/bin/sh
echo ”Content-type: text/html”
echo ””
paiva=`date`
cat << EndOfHeader
<HTML><BODY bgcolor=’#ffffff’>
<P><CENTER><B>
<FONT color=’#ff0000’ size=’5’>
Tänään on<BR>
$paiva
</B></CENTER></BODY></HTML>
EndOfHeader
```

Tuloste voidaan ohjata tiedostoon (>) -merkillä seuraavasti.

Esimerkki.

```
cat << EndOfHeader > koe.txt
Tämä menee nyt
tiedostoon
nimeltä
koe.txt
EndOfHeader
```

Putkimekanismilla voidaan komennon tuloste ohjata syötteeksi toiselle komennolle (|) -merkkiä käyttämällä. Lähetetään esimerkiksi yllä tehty tiedosto sähköpostilla vastaanottajalle.

```
cat koe.txt|mail tunnus@kone.domain.fi
```

Ncsa-palvelimen mukana seuraa yleensä muutama valmis scripti, joiden joukossa on mm. **date**, **mail** ja **finger** -nimiset scriptit. Näiden nimet pitää muuttaa, jos halutaan käyttää samannimisiä käyttöjärjestelmäkomentoja samassa hakemistossa.

Myös ehtolauseet ovat mahdollisia kuten **if**-lause. Unix-komennoilla on yleensä myös paluuarvo, joka on nolla jos komennon suoritus onnistui tai nollasta poikkeava arvo jos komennon suoritus epäonnistui. Sh-komentotulkki tallettaa paluuarvon muuttujaan **$?**, mutta sitä voi myös ehtolauseissa usein käyttää suoraan. If-lause on seuraavaa muotoa.

if *ehto*
then
komennot
[else]
komennot
fi

HUOM! Paluuarvo on siis 0, jos ehto on tosi.

Esimerkki. Tulostaa tekstin ” hakemiston luonti onnistui jos **mkdir**-komento onnistui ja tekstin ” hakemiston luonti ei onnistunut” jos mkdir -komento ei onnistunut. Lisäksi lopetaan tiedoston ajo komennolla **exit**. Luku 1 ilmaisee, että lopetus johtui virhetilanteesta (luku 0 ilmaisisi kaikki hyvin tilannetta).

```
#!/bin/sh
if mkdir koe
then
echo hakemiston luonti onnistui
else
echo hakemiston luonti ei onnistunut
exit 1
fi
```

Mukana voidaan käyttää komentoja **true**, joka palauttaa aina arvon tosi ja **false**, joka palauttaa aina arvon epätosi. Lisäksi voidaan käyttää komentoa **test** yhdessä **if**-lauseen kanssa. Test-komennon paluuarvo on 0

(tosi) tai 1 (epätosi) riippuen siitä onko argumenttina annettu ehto tosi vai epätosi.

Esimerkki. Testataan onko tiedosto olemassa.

```
#!/bin/sh
if test -f tiedosto_nimi
then
echo tiedosto on olemassa
fi
```

Test-komennon kirjoitus voidaan korvata ”**[** ” ja” **]**” -merkeillä (Huom! Välilyönnit). Esimerkiksi edellinen komento kirjoitettaisiin silloin

```
if [ -f tiedosto_nimi ]
```

Merkkijonojen yhtäläisyys testataan muodolla.

```
if [ merkkijono_1 = merkkijono_2 ]
```

Lukuja vertaillaan muodolla (tässä yhtäsuuruutta).

```
if [ luku_1 -eq luku_2 ]
```

Yhteenveto operaattoreista:

Ehto	**Tosi, jos**
luku_1 -eq luku_2	luvut ovat yhtäsuuret
luku_1 -ne luku_2	luku_1 eri suuri kuin luku_2
luku_1 -gt luku_2	luku_1 > luku_2
luku_1 -ge luku_2	luku_1 >= luku_2
luku_1 -lt luku_2	luku_1 < luku_2
luku_1 -le luku_2	luku_1 <= luku_2
mj_1 = mj_2	merkkijonot yhtäläiset
mj_1 != mj_2	merkkijonot erilaiset
mj	merkkijono ei ole tyhjä
-n mj	merkkijono (mj) ei ole tyhjä
-z mj	merkkijono (mj) on tyhjä
-r tiedosto_nimi	tiedosto on luettavissa
-w tiedosto_nimi	Käyttäjällä on kirjoitusoikeus
-f tiedosto_nimi	tiedosto on tavallinen tiedosto
-d tiedosto_nimi	tiedosto on hakemisto
-s tiedosto_nimi	tiedosto on olemassa ja sen koko on suurempi kuin nolla tavua

---

Myös silmukoita voidaan käyttää ja niitä on kahta eri tyyppiä eli tyypit **for** ja **while**. For silmukka antaa muuttujalle vuorotellen kaikki arvot listasta ja suorittaa joka kerran samat komennot.

for *muuttuja* in *lista*
do *komennot*
done

Esimerkki. Tulostetaan listassa olevat luvut.

```
#!/bin/sh
for luku in 2 4 6 8
do echo $luku
done
```

While-silmukkaa suoritetaan kunnes ehto saa arvon epätosi.

while *ehto*
do *komennot*
done

Vielä löytyy yksi lause eli **case**-lause, jolla testataan merkkijonon yhtäläisyyttä vaihtoehtoisiin lausekkeisiin. Jos testaus onnistuu vaihtoehtoa vastaava komentojono suoritetaan. Lausekkeita ja komentolistoja voi olla niin monta kuin tarvitaan.

case *merkkijono* in
lauseke_1)
komennot_1
;;
lauseke_2)
komennot_2
;;
...
esac

Esimerkki.

```
#!/bin/sh
ehto=uusi
case  $ehto in
vanha )
echo Yhteystiedot ovat vanhentuneet
;;
uusi )
echo Yhteystiedot ovat ajan tasalla
;;
esac
```

5.2.3 Komentotiedostojen parametrit

Komentotiedostoille voidaan välittää 9 parametriä, **joita merkitään $0 - $9** , joista $0 :ssa on itse komentotiedoston nimi ja näin ollen $1 on ensimmäinen parametri, $2 toinen ja niin edelleen. Parametrien lukumäärän voi lukea muuttujasta **$#**. Jos halutaan käyttää kaikkia parametrejä ne on luettavissa yhtenä listana muuttujasta **$*** tai yhtenä merkkijonona muuttujasta **$*** .

Esimerkki. Tulostetaan ensin kaikki parametrit yksitellen ja sitten kaikki kerralla.

```
#!/bin/sh
for param in $*
do
echo $param
done
echo "$*"
```

5.2.4 Parametrien välitys CGI-ohjelmaan URL:n avulla

Parametrit välitetään CGI-ohjelmalle URL:ssä kysymysmerkin (?) avulla. Esimerkiksi ohjelma *tulosta* saa parametrikseen merkkijonon ”koe”.

http://www.kone.fi/cgi-bin/tulosta?koe

Jos parametrejä on enemmän kuin yksi ne erotetaan toisistaan (+) -merkillä. Esimerkiksi ohjelma *tulosta* saa parametreikseen merkkijonot ”*kokeilen*” ”*ohjelmaa*”.

http://www.kone.fi/cgi-bin/tulosta?kokeilen+ohjelmaa

Myös lähetystapa GET lisää lomakkeen sisällön (nimi=arvo -parit) parametriksi kysymysmerkin jälkeen yhteen merkkijonoon.

Yllä kuvatulla tavalla välitetyt parametrit ovat luettavissa **$1 - $9** -muuttujista kuten esitettiin kappaleessa 6.2.3 *komentotiedostojen parametrit*.

C- ja C++ -kielissä parametrit luetaan muuttujien argc:n ja argv:n avulla.

5.3 CGI-ohjelmien ympäristömuuttujat

CGI-ohjelmille voidaan välittää käyttökelpoista tietoa käytössä olevasta ympäristöstä ympäristömuuttujien avulla. Ympäristömuuttujien nimet kirjoitetaan isoilla kirjaimilla ja käytettäessä unix-komentotiedostoissa, lisätään nimen eteen dollari ($) -merkki. Muilla kielillä kirjoitetuissa ohjelmissa pitää ympäristömuuttujien lukeminen tarkistaa asianomaisesta manuaalista. Seuraavassa luetellaan joukko CGI-ohjelmien ympäristömuuttujia.

AUTH_TYPE

Suojattua ohjelmaa käynnistettäessä käytetty tunnistusmetodi.

CONTENT_LENGTH

Standard input (stdin) -puskurissa olevan tiedon määrä tavuina. Tästä ympäristömuuttujasta löytyy stdin-puskurissa olevan merkkijonon pituus käytettäessä lähetystapaa POST lomakkeiden yhteydessä, koska yksi merkki esitetään yhdellä tavulla (8 bittiä).

CONTENT_TYPE

Tämä ympäristömuuttuja sisältää palvelinkutsussa käytetyn MIME-tyypin, jos on käytetty metodia POST.

GATEWAY_INTERFACE

Sisältää palvelimen käytössä olevan CGI-version muodossa CGI/*versio*.

HTTP_ACCEPT

Sisältää tiedon lukuohjelmalle kelpaavista MIME-tyypeistä. Eri MIME-tyypit on eroteltu pilkulla (,) toisistaan.

HTTP_FROM

Sisältää palvelupyynnön tekevän käyttäjän sähköpostiosoitteen. (Useimmat palvelimet eivät tue tätä ominaisuutta).

HTTP_*otsikkokenttä*

Sisältää itsemääritellyn otsikkokentän arvon. Itse otsikkokentän nimi tulee tekstin ”*otsikkokenttä*” paikalle.

HTTP_REFERER

Sisältää sen dokumentin URL-osoitteen, johon palvelinohjelma osoitti kutsuessaan CGI-ohjelmaa.

HTTP_USER_AGENT

Sisältää lukuohjelman tyypin muodossa *ohjelman_nimi/versiokirjasto/versio*.

PATH_INFO

Sisältää polkuinformaation, joka välitetään CGI-ohjelmalle.
PATH_TRANSLATED

Sisältää käännetyn version PATH_INFO -muuttujan sisällöstä.

QUERY_STRING

Sisältää ohjelmalle URL:ssä annetut parametrit (kaikki ? -merkin jälkeiset merkit) yhtenä merkkijonona. Myös mahdolliset (+) -merkit ovat mukana. Lomakkeiden yhteydessä käytettäessä lähetystapaa GET, löytyy lomakkeen sisältö (nimi=arvo -parit) tästä ympäristömuuttujasta.

REMOTE_ADDR

Sisältää palveltavan lukuohjelman IP-numeron (esimerkiksi 193.166.41.1).

REMOTE_HOST

Sisältää palveltavan lukuohjelman IP-osoitteen (esimerkiksi impedanssi.it.spt.fi).

REMOTE_IDENT

Sisältää lukijan käyttäjätunnuksen, jos palvelin tukee käyttäjätunnuksen varmistusta.

REMOTE_USER

Sisältää lukijan käyttäjätunnuksen, jos ohjelma oli suojattu salasanalla. HUOM! Tämä ei ole (välttämättä) sama käyttäjätunnus kuin lukijan palvelinkoneessa oleva.

REQUEST_METHOD

Sisältää palvelua pyydettäessä käytetyn metodin. HTML-protokollalla esimerkiksi GET tai POST.
SCRIPT_NAME

Sisältää ajettavan ohjelman sijainnin.

SERVER_NAME

Sisältää ohjelman käynnistäneen WWW-palvelimen IP-osoitteen.

SERVER_PORT

Sisältää WWW-palvelimen käyttämän portin.

SERVER_PROTOCOL

Sisältää protokollan, jolla ohjelma kutsuttiin muodossa *protokolla/versio*.

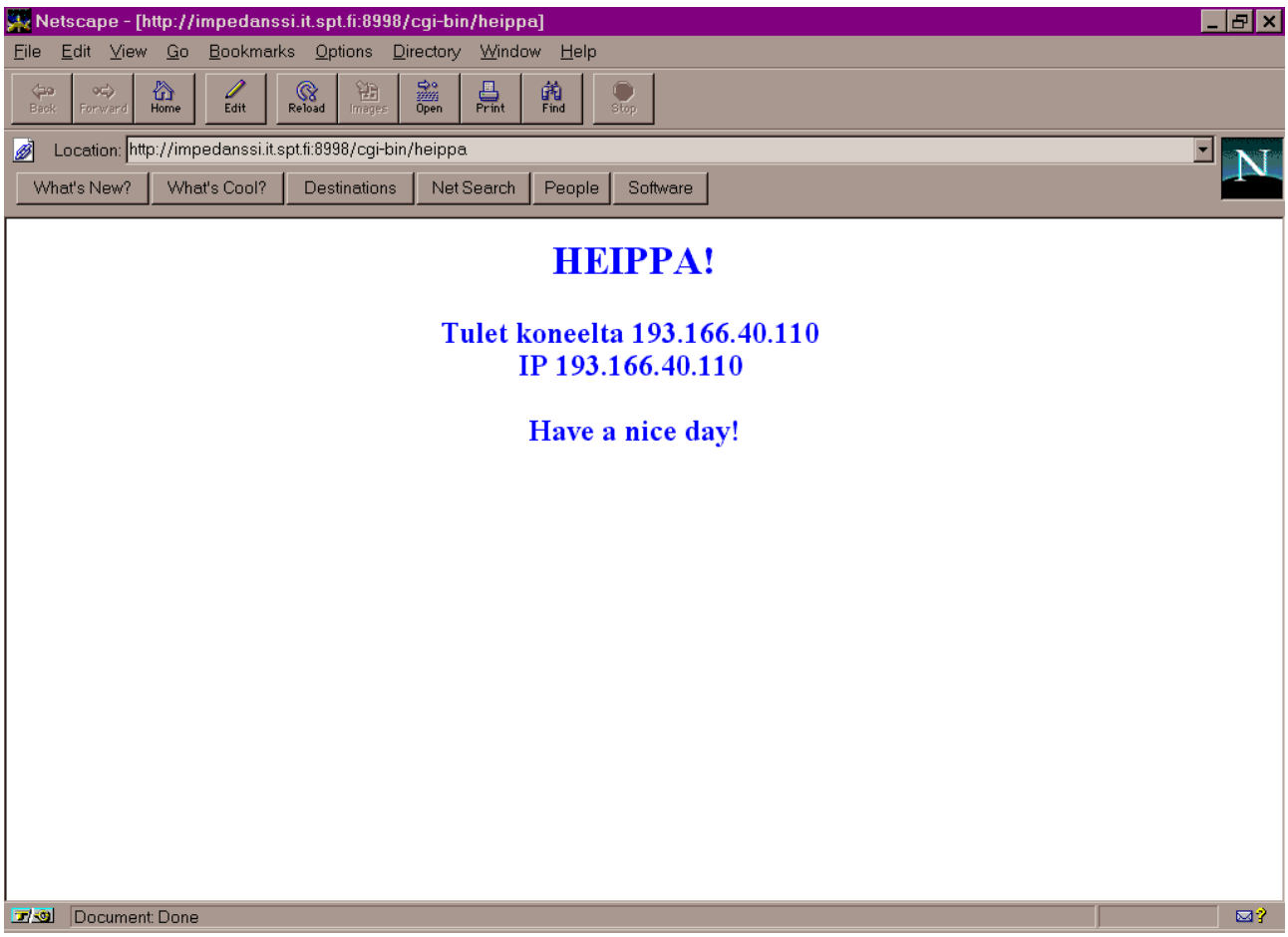
SERVER_SOFTWARE

Sisältää ohjelman käynnistäneen palvelinohjelmiston nimen ja version muodossa *nimi/versio*.

Esimerkki. Tulostetaan lukuohjelmaa käyttävän koneen IP-osoite (jos saadaan selville) ja IP-numero.

```
#!/bin/sh
echo  "Content-type: text/html"
echo  ""
echo  "<BODY BGCOLOR='#FFFFFF'>"
echo  "<FONT size='5' color='#0000FF'>"
echo  "<P><CENTER><B><BIG>”
echo  ”HEIPPA!</BIG></B><BR>"
echo  "<BR><B> Tulet koneelta "
echo $REMOTE_HOST
echo  "<BR> IP "
echo $REMOTE_ADDR
echo "<BR><BR> Have a nice”
```

```
        echo "day!</B><BR><BR></CENTER> "
        echo "</BODY></HTML>"
```



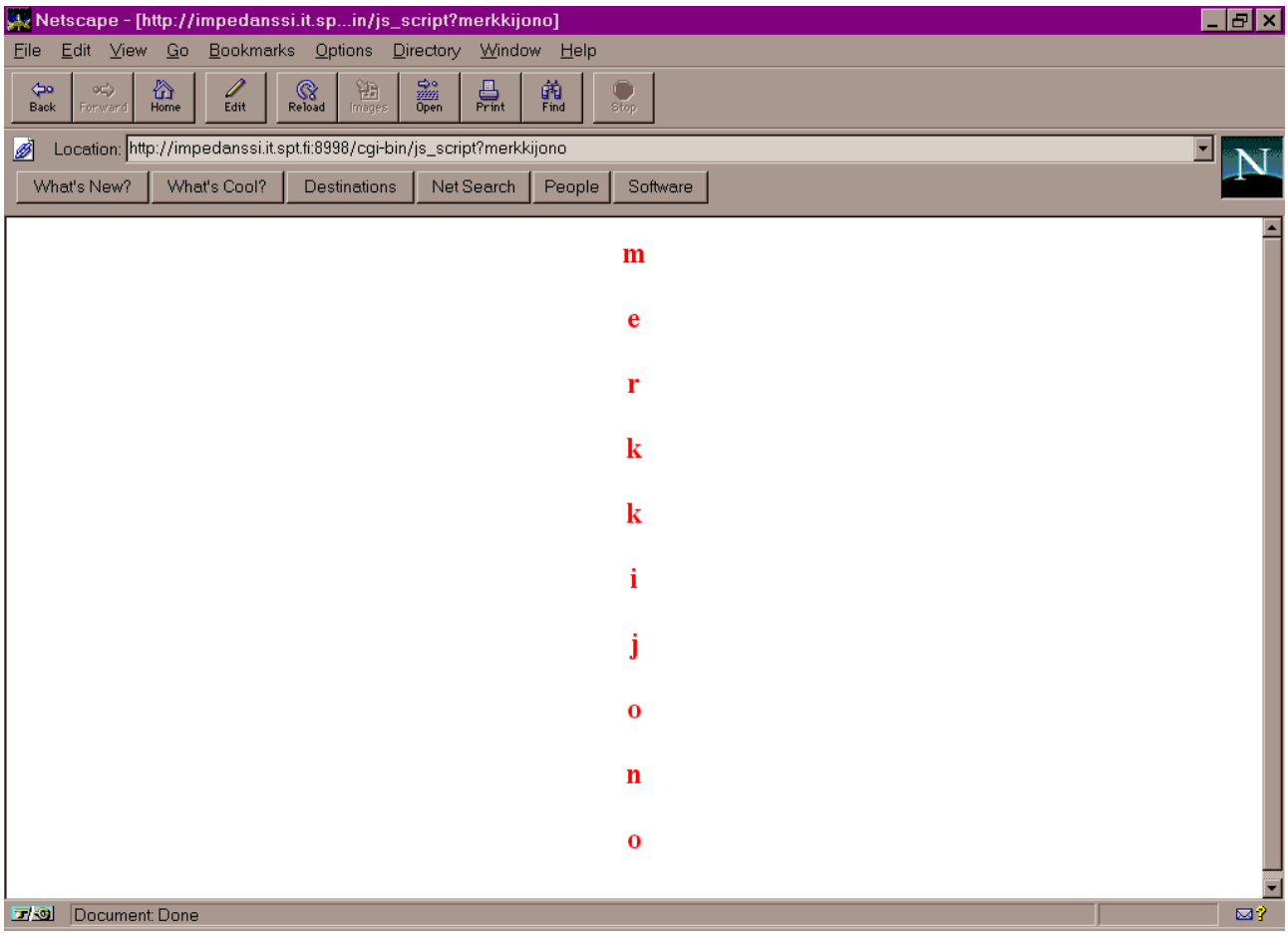
Edellisen esimerkin tuloste. Myös osoitteen paikalla näkyy IP-numero, koska varsinaista nimeä ei lukuohjelmaa ajavalle koneelle ole annettu.

5.3.1 Javascript komentotiedostoissa

Komentotiedostoissa voi käyttää myös JavaScript-kieltä HTML-kielen lisäksi. Komentotiedoston ensimmäiset rivit pidetään ennallaan ja sen jälkeen voidaan aloittaa JavaScript-koodin kirjoitus yhdessä HTML-koodin kanssa.

Esimerkki. Tulostetaan lukuohjelmaan parametrinä annettu merkkijono pystysuunnassa ja keskitettynä.

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
cat << EndOfHeader
<HTML>
<BODY bgcolor='#ffffff'>

<SCRIPT LANGUAGE='JavaScript'>


var m_jono='$QUERY_STRING' ;   //Luetaan parametrin arvo



for(var i=0;i<m_jono.length;i++)

        {

        merkki=m_jono.charAt(i);

        document.write('<CENTER><B>',merkki.fontcolor('#f
f0000')                 .fontsize(5).blink());

        document.write('</B></CENTER><BR>');

        }


</SCRIPT>
</BODY>
</HTML>
EndOfHeader
```



Edellisen esimerkin tuloste, kun parametriksi annettiin ”merkkijono”.

5.4 Lomakkeiden käsittely

Lomakkeiden sisältö lähetetään yhtenä merkkijonona lähetystavasta riippuen eri kanavia pitkin ACTION-parametrilla määrättyyn osoitteeseen (*URL*). Lähetettävä merkkijono koostuu nimi=arvo -pareista, jotka on erotettu toisistaan **&**-merkillä. Merkkijono on usein sellaisenaan vaikeaselkoinen ja näin ollen sitä pitää muokata ennen kuin ryhdytään lisätoimenpiteisiin (esimerkiksi lähetetään lomakkeen vastaukset sähköpostilla vastaanottajalle). Oletetaan, että on luotu lomake, jossa on kolme tekstikenttää (*text-oliota*), jotka on nimetty *eka*, *toka* ja *kolmas*. Oletetaan lisäksi, että käyttäjä kirjoittaa kenttiin tekstit ”*ensimmäinen*”, ”*toinen*” ja ”*kolmas*”. Kun tämä lomake lähetetään vastaanottavalle ohjelmalle submit-oliolla, se saa käsiteltäväkseen seuraavan merkkijonon.

```
eka=ensimmäinen&toka=toinen&kolmas=kolmas
```

Käsittelevän ohjelman tulee purkaa tämä merkkijono ja suorittaa sen jälkeen mahdolliset jatkotoimenpiteet.

Huom! Ääkköset ja monet erikoismerkit lähetetään heksakoodattuna ja ne on koodattava auki ennen jatkokäyttöä. Merkkien koodi muodostetaan merkin ASCII-koodista (heksana) ja %-merkistä, joka lisätään koodiluvun eteen. Esimerkiksi pieni ä-kirjain koodataan **%**E4. Lisäksi tyhjät välilyönnit lähetetään **+** -merkkeinä.

Seuraavassa koodattuja merkkejä ja niiden koodeja.

ä=%E4	Ä=%C5	Ö=%D6	ö=%F6	å=%E5	Å=%C5
/=%2F	(=%28	)=%29	#=%23	¤=%A4	[=%5B
]=%5D	~=%7E	'=%27	§=%A7	½=%BD	?=%3F
!=%21	,=%2C	;=%3B	:=%3A	$=%24	|=%7C
<=%3C	>=%3E	\=%5C	{=%7B	}=%7D	£=%A3

5.4.1 Lähetystapa GET

Käytettäessä lähetystapaa GET vastaanottava ohjelma saa lomakkeen sisällön yhtenä merkkijonona parametrikseen. Sisältö on luettavissa ympäristömuuttujasta QUERY_STRING. Lähetystavalla GET ei voida lähettää kovin pitkiä lomakkeita, lisäksi sisältö näkyy (mahdollisine salasanoineen) URL:ssä. Lomakkeen purku voidaan hoitaa vaikka JavaScriptiä hyödyntämällä seuraavasti. (purkuosan jälkeen kirjoitetaan tapauskohtainen koodi).

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
cat << EndOfHeader
```

```
<HTML>
<HEAD>
<!-- Juha Niemi 8.9.1996 -->
<SCRIPT LANGUAGE='JavaScript'>

function tee_taulu(n){ //Luodaan taulukkomuuttuja

this.length=n;

for(var j=1; j<n; j++){

this[j]=0;}

return this;
}



//Heksakoodattujen merkkien etsintä ja korjaus

//ÄLÄ KÄYTÄ YLIMÄÄRÄISIÄ (&) - TAI (=) -MERKKEJÄ

//Välilyönti tulee + -merkkinä, purku suoraan ohjelmakoo-
diosassa


function avaa_koodi(merkki){

var paluu;

merkit=new tee_taulu(32);      //Luodaan taulu ja anne-
taan muuttujille arvot


merkit[1]='%E4=ä';     // ä
merkit[2]='%C5=Ä';     // Ä
merkit[3]='%D6=Ö';     // Ö
merkit[4]='%F6=ö';     // ö
merkit[5]='%E5=å';     // å
merkit[6]='%C5=Å';     // Å
merkit[7]='%2F=/';             // /
merkit[8]='%28=(';             // (
merkit[9]='%29=)';             // )
merkit[10]='%A4=¤';    // ¤
merkit[11]='%5B=[';    // [
merkit[12]='%5D=]';    // ]
merkit[13]='%23=#';    // #
```

```
merkit[14]='%7E=~';     // ~
merkit[15]='%A7=§';     // §
merkit[16]='%BD=½';     // ½
merkit[17]='%3F=?';     // ?
merkit[18]='%21=!';     // !
merkit[19]='%2C=,';     // ,
merkit[20]='%3B=;';     // ;
merkit[21]='%3A=:';     // :
merkit[22]='%24=$';     // $
merkit[23]='%7C=|';     // |
merkit[24]='%3C=<';     // <
merkit[25]='%3E=>';     // >
merkit[26]='%5C=/';     // \
merkit[27]='%7B={';     // {
merkit[28]='%7D=}';     // }
merkit[29]='%A3=£';     // £
merkit[30]='%0A=<BR>'; // rivinvaihto <BR>
merkit[31]="%27='";     // '


for(var i=1;i<32;i++)
        {

        if(merkki==merkit[i].substring(0,3))
//Etsitään koodi
                {

                if(i!=30)
                paluu=merkit[i].charAt(4);
//Puretaan koodi
                if(i==30)
                paluu=merkit[i].substring(4,8);
                //Puretaan koodi, jos se oli rivinvaihto

                }
        }

 return paluu;
}

</SCRIPT>
<!--  -->
</HEAD>
<BODY bgcolor='#ffffff'>
<SCRIPT LANGUAGE='JavaScript'>

var m_jono='$QUERY_STRING' ;
```

```
var nimi="";           //Nimikentän arvo
var arvo="";           //Arvokentän arvo
var laskin=2;          //Parillinen/pariton laskin
var parimaara=0;               //nimi=arvo -parien luku-
määrä
var i=0;        //indeksi
var j=1;        //indeksi
var et_merkit;         //& -merkkilaskuri
var param='';          //Parametri välitettäväksi
avaa_koodi funktiolle
var avattu='';          //Auki koodattu merkki
var koodattu=false;    //koodattu vai ei




for(i=0;i<m_jono.length;i++)
{
et_merkit=m_jono.charAt(i);  //Tutkitaan &-merkkien luku-
määrä
if(et_merkit=='&')parimaara++; //lasketaan nimi-arvo -
parien lukumäärä
}

parimaara++;   //Lisätään viimeinen pari
muuttujat= new tee_taulu(2*parimaara+1);
 //Luodaan taulukkomuuttuja


for(i=0;i<m_jono.length;i++)
//Puretaan merkkijono nimi- ja arvokentiksi
       {


if(m_jono.charAt(i)!= '=' && m_jono.charAt(i)!='&' &&
laskin%2==0)
       {

       if(m_jono.charAt(i)=='+')
//Välilyönnin palautus
              {
              avattu=' ';
              koodattu=true;
              }
```

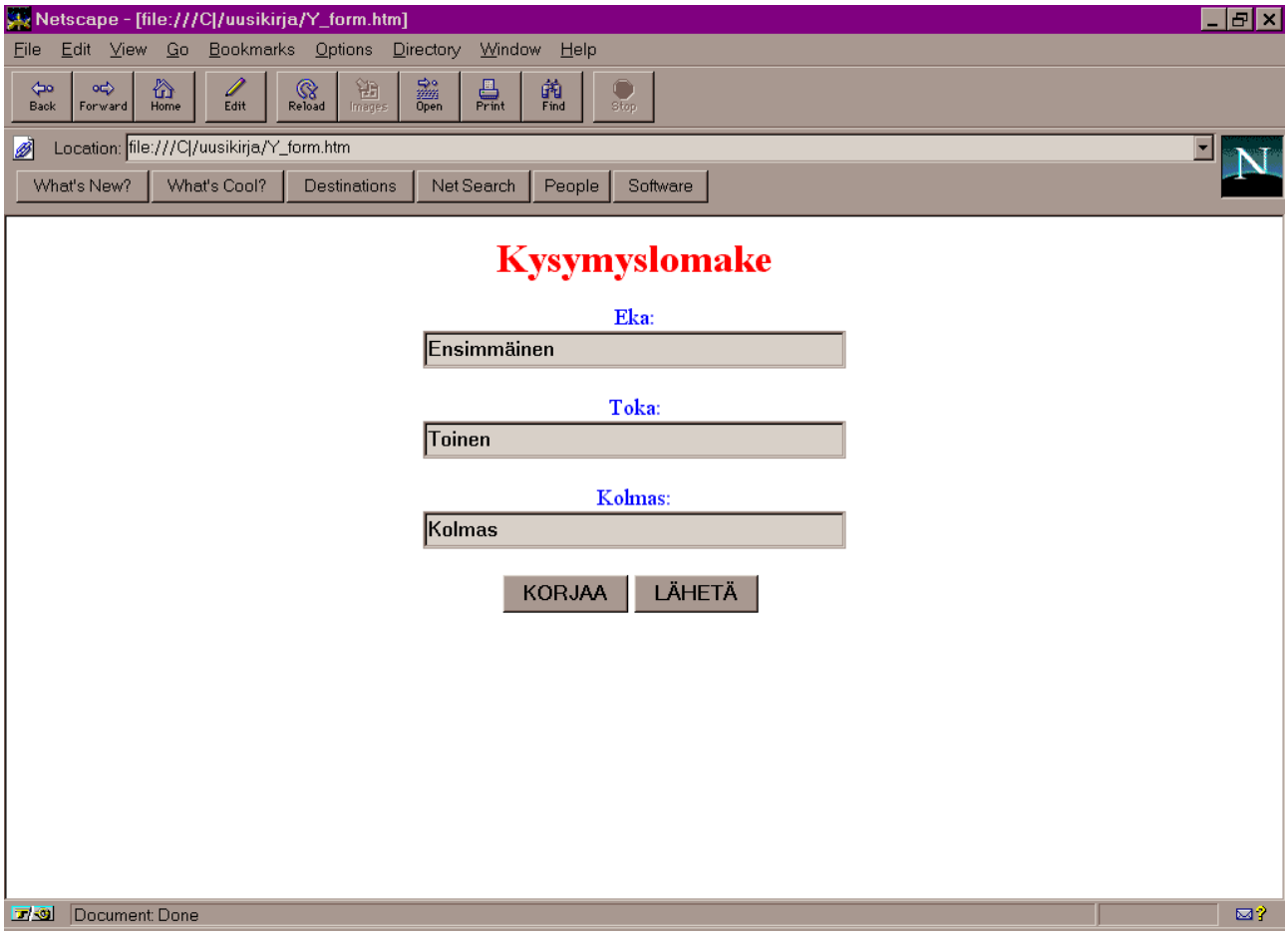
```
        if(m_jono.charAt(i)== '%')
//Onko koodattuja merkkejä
                {
                param=m_jono.substring(i,i+3);
//Välitetään koodi
                avattu=avaa_koodi(param);
                koodattu = true;
                i+=2;           //indeksin päivitys
                }

        nimi+=(koodattu)?avattu:m_jono.charAt(i);
        koodattu=false;
        }



        if(m_jono.charAt(i)=='=')
        {
        muuttujat[j]=nimi;
        nimi="";                //Tyhjennetään muuttuja
        j++;
        }



if(m_jono.charAt(i)!='=' && m_jono.charAt(i)!='&' && las-
kin%2!=0)
        {

        if(m_jono.charAt(i)=='+')
//Välilyönnin palautus
        {
        avattu=' ';
        koodattu=true;
        }


        if(m_jono.charAt(i)== '%')
//Onko koodattuja merkkejä
                {
                param=m_jono.substring(i,i+3);
//Välitetään koodi
                avattu=avaa_koodi(param);
                koodattu = true;
                i+=2;           //indeksin päivitys
                }
```

```
        arvo+=(koodattu)?avattu:m_jono.charAt(i);
        koodattu=false;
        }



        if(m_jono.charAt(i)=='&')
        {
        muuttujat[j]=arvo;
        arvo="";                //Tyhjennetään muuttuja
        j++;
        }

        if(m_jono.charAt(i)=='=' ||
m_jono.charAt(i)=='&')laskin++;

}

muuttujat[j]=arvo;
//Otetaan viimeinen arvo talteen



//Tästä eteenpäin tapauskohtainen koodi


for(i=1;i<muuttujat.length;i++)//Muuttujien tulostus
{

if(i%2!=0)
{
document.write('<BR>Nimikenttä = ');
document.write(muuttujat[i]);
}

if(i%2==0)
{
document.write('<BR>Arvokenttä = ');
document.write(muuttujat[i],'<BR><BR>');
}

}

</SCRIPT>
</BODY>
</HTML>
```

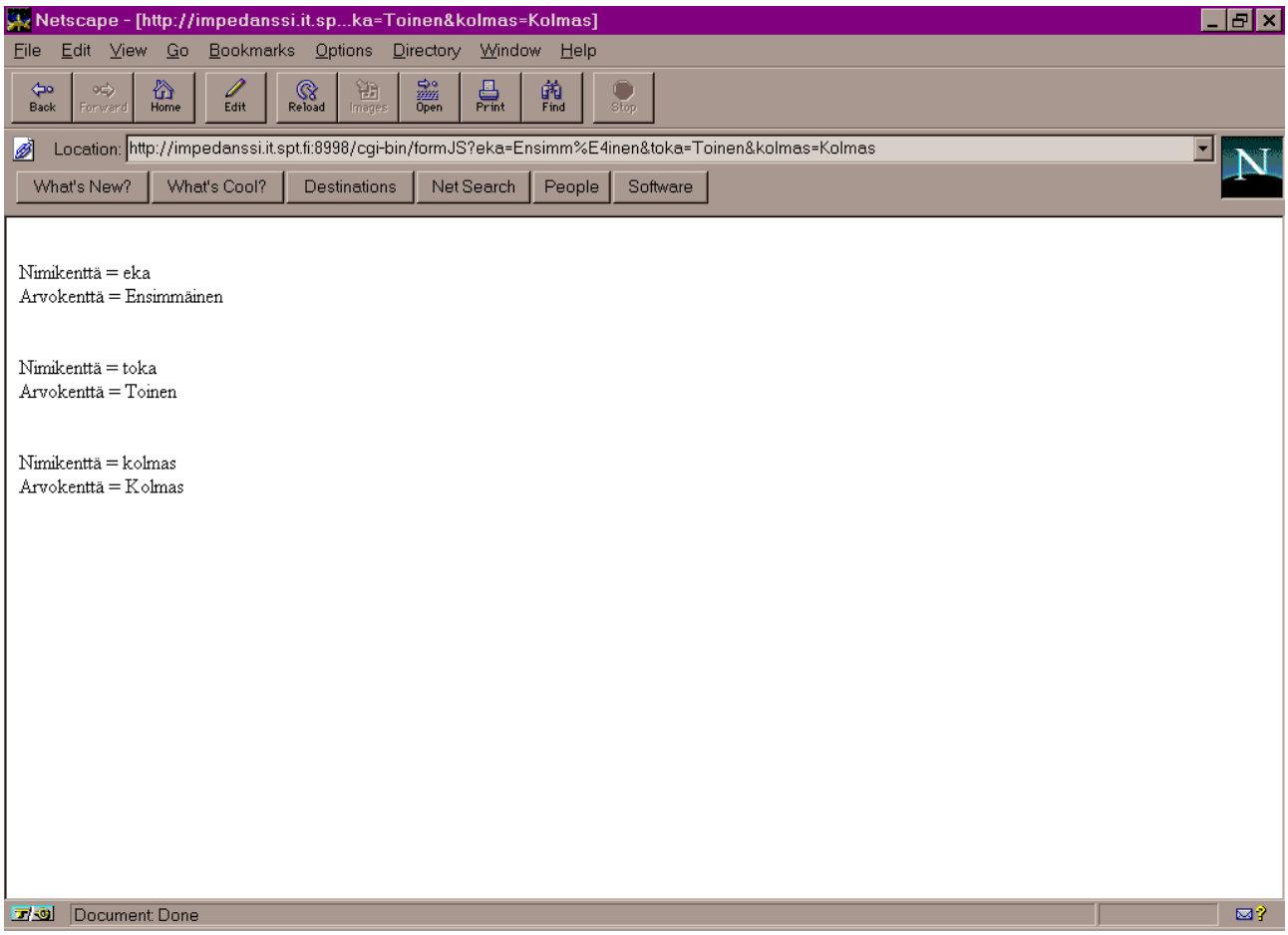
```
EndOfHeader
```

Esimerkki. Luodaan seuraava lomake ja lähetetään se edelliselle JavaScriptiä hyödyntävälle CGI-ohjelmalle, joka purkaa sen.

```
<HTML>
<BODY bgcolor="#ffffff">

<CENTER>
<font color="#ff0000">
<H1>Kysymyslomake</H1>
</CENTER>

<FORM  method="get"
ACTION="http://impedanssi.it.spt.fi:8998/cgi-bin/formJS">


<font color="#0000ff" size="4">

<CENTER>

Eka:<BR>
<INPUT TYPE="text" NAME="eka" SIZE="40"><BR><BR>

Toka:<BR>
<INPUT TYPE="text" NAME="toka" SIZE="40"><BR><BR>

Kolmas:<BR>
<INPUT TYPE="text" NAME="kolmas" SIZE="40"><BR><BR>

<INPUT TYPE="reset" VALUE="KORJAA">
<INPUT TYPE="submit" VALUE="LÄHETÄ">

</CENTER>
</FORM>

</BODY>
</HTML>
```



Edellisen esimerkin lomake, kun kenttiin on kirjoitettu vastaukset. Lähetä -nappia ei ole vielä painettu. Kun lähetä-nappia painetaan lomakkeen sisältö lähetetään parametrinä ACTION-parametrillä määrättyyn osoitteeseen (*URL*), joka tässä johtaa ohjelmaan *formJS*.



Edellisen esimerkin tuloste, kun lomake on lähetetty sen sisällön purkavalle CGI-ohjelmalle, *formJS* (JavaScriptiä hyödyntävä unix-komentotiedosto). Ohjelman koodi esiteltiin jo aikaisemmin.

5.4.2 Lähetystapa POST

Kun käytetään lähetystapaa POST, käsittelevä ohjelma voi lukea lomakkeen sisällön **stdin**-puskurista yhtenä merkkijonona. Form-tagissa on määrättävä lähetystavaksi POST **method**-parametrillä (METHOD=”POST”). Huom! Lähetystavan oletusarvo on GET, jos mitään arvoa ei määritellä. Stdin-puskurissa olevan merkkijonon pituus on luettavissa ympäristömuuttujasta CONTENT_LENGTH.

Verkosta on saatavissa **Un-CGI** -niminen ohjelma, joka purkaa POST-lähetystavalla lähetetyn lomakkeen sisällön ympäristömuuttujiin. Ympäristömuuttujien nimet ovat muotoa WWW_*nimi,* jossa nimen paikalle

tulee lomakkeen kentän nimiosa ja itse muuttujan arvo on lomakkeen kentän arvo. Ohjelma löytyy seuraavasta osoitteesta.

http://www.hyperion.com/~koreth/uncgi.html

Lomakkeen purkuohjelman, joka käyttää lähetystapaa POST, voi tehdä myös itse melko vähällä vaivalla. Tehdään ohjelma joka purkaa lomakkeen sisällön tiedostoon nimi=arvo -pareiksi riveittäin ja tämän jälkeen tehdään toinen ohjelma, joka lukee tätä tiedostoa halutulta riviltä. Tiedostoa lukevaa ohjelmaa käytetään komentotiedostoista, joihin kirjoitetaan kaikki muotoilu- ja mahdolliset muut jatkotoimenpiteet.

Ensimmäiseksi tehdään **cgi-bin** -hakemistoon alihakemisto **data** komennolla **mkdir data** ja tämän jälkeen annetaan sille kaikki oikeudet komennolla **chmod 777 data**. Tämä sen vuoksi, että itse palvelinohjelmalla on yleensä oma käyttäjä- ja ryhmätunnuksensa, joka poikkeaa lomakkeen purkuohjelman vastaavista ja lisäksi cgi-bin -hakemiston omistaja voi olla joku muu. Data-hakemistolla (jossa on kaikki oikeudet) varmistetaan se, että lomakkeen purkuohjelma pystyy kirjoittamaan levylle. Data-hakemistossa sijaitsevien tiedostojen oikeuksia voidaan sitten jo haluttaessa rajata. Seuraava ohjelma (form) kirjoittaa nimi=arvo -parit tiedostoon **data/arvoparit.dat**. Tiedosto on tavallinen tekstitiedosto ja nimi=arvo -parit on luettavissa sieltä. Uudet arvot hävittävät vanhat. Seuraavassa ohjelman koodi (ohjelma on kirjoitettu C++ -kielellä).

Lähdekoodi: form.cpp

```
/* WWW-lomakkeen käsittelyohjelma versio 2.3 Juha Niemi
8.9.1996  */

/*Ohjelma purkaa lomakkeelta tulevan merkkijonon ni-
mi=arvo -pareiksi ja tallettaa
  ne tiedostoon data/arvoparit.dat scriptien tai muiden
ohjelmien käytettäväksi,
  Tiedoston voi lukea ohjelmalla get_f . joka saa para-
metrin kautta tiedon
  siitä mikä kenttä (nimi tai arvo) luetaan ja miltä ri-
viltä tiedostosta */
```

```
/* arvoparit.dat tiedosto (tekstitiedosto) säilyy sellai-
senaan kunnes uusi ajo muuttaa sitä
   ja on luettavissa (ja muutettavissa editorilla)  */

//Hakemistoon (cgi-bin), jossa ohjelma sijaitsee pitää
tehdä data - niminen alihakemisto

/*data -hakemistolle on hyvä antaa kaikki oikeudet komen-
nolla chmod 777 data
  Näin välttyy oikeuksien puuttumisesta mahdollisesti ai-
heutuvilta ongelmilta */


/* Ajettavat scriptit ja ohjelmat voi sijoittaa myös sen
hakemiston alihakemistoon, jossa
        form -ohjelma sijaitsee, tällöin pitää siinäkin
hakemistossa olla alihakemisto nimeltä
        data. Ajettava ohjelma välitetään silloin seuraa-
vasti.
        http://www.kone.fi/cgi-
bin/form?hakemistopolku/ohjelma  */


//Ohjelmaa koodaa auki myös erikoismerkit, jotka lukuoh-
jelma lähettää heksakoodattuna


/*Ohjelman kutsu lomakkeen ACTION-parametrissa

  <FORM METHOD="POST" ACTION="http://www.kone.fi/cgi-
bin/form?nimi">

 nimen paikalle suoritettavan scriptin tai ohjelman nimi,
jossa kerrotaan mitkä
 nimi=arvo -parit luetaan get_f-ohjelman avulla. Ohje pa-
rametrien käytöstä get_f ohjelman
 lähdekoodissa  */



// Jos parametri puuttuu, suoritettavan scriptin (tai oh-
jelman) oletusnimeksi tulee cgi_script


//Esimerkki scriptin käytöstä: Katso get_f lähdekoodi
```

```
// Lomakkeelta tuleva tieto on muotoa:    ni-
mi=arvo&nimi=arvo&nimi=arvo



//Tieto on lähetettävä POST-metodilla, jolloin se on lu-
ettavissa stdin-puskurista

//stdin-puskuriin saapuvan merkkijonon pituus on luetta-
vissa ympäristömuuttujasta CONTENT_LENGTH


/*Ohjelmalle voi antaa myös toisen parametrin eli merkki-
jonon DEBUG (vain isoilla kirjaimilla)
  seuraavasti.

Toinen parametri on DEBUG ->  Ohjelma ajaa ensin ensim-
mäisessä parametrissa annetun scriptin ja tulostaa sen
tulosteen perään stdin -puskurin sisällön, nimi=arvo -
parien lukumäärän ja nimi=arvo -parit samalle www-
sivulle.
 (esim. http://www.kone.fi/cgi-bin/form?scripti+DEBUG)
HUOM! Ajettavassa scriptissä tai  ohjelmassa täytyy olla
seuraavat rivit ennen HTML-tulostusta (muuten seurauksena
on server error ilmoitus www-sivulle).


          #!/bin/sh
          echo "Content-type: text/html"
          echo ""

          Tai

          cout<<"Content-type: text/html\n";
          cout<<"\n";

          Tai

          printf("Content-type: text/html\n");
          printf("\n");        */

/*   Jos seuraava virheilmoitus ilmestyy lukuohjelmaan


500 Server Error
```

```
The server encountered an internal error or misconfigura-
tion and was unable to complete your request.

Please contact the server administrator, emiju-
ha@jazz.cc.spt.fi and inform them of the time the error
occurred , and anything you might have done that may have
caused the error.

Error: HTTPd: malformed header from script
/home/emijuha/www/httpd_1.5a/cgi-bin/form


Tarkista :

1. Tiedostojen ja hakemistojen oikeudet (sopiva chmod 755
nimi) riittämättömät.
2. HTML-header (Content-type: text/html ja seuraava rivi
tyhjä) scriptissä
puuttuu, on virheellinen tai puutteellinen. ^M -merkit
voivat vaikuttaa tähän.
3. Ohjelma käynnistetty suoraan URL:stä tai väärä lähe-
tysmetodi (GET: pitää olla POST)-> stdin puskuri on tyh-
jä.

*/



//Nimikentän merkkien lukumäärä on rajoitettu: max 15

//Arvokentän merkkien lukumäärä on rajoitettu: max 250









#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

```
class form {

        struct lomake {char nimi[16];char arvo[251];};
        lomake *tp;
        int ko-
ko,n,m,check_n,check_a,check_p,laskin,nlaskin,x;
        char
*syote,apu_mr,*virhe,*test_mr,*chm,*chp,*tulos;


public:
        lomake p[50];
        int pmaara;
        form();
        ~form(){ delete tp;delete syote;delete vir-
        he;delete tulos;delete test_mr;}
        void lue_stdin();
        void kasittele();
        void tallenna();
        void print();
        void printhtml_script();
};


form::form(){

  tp=new lomake;
  virhe=new char[44];
  test_mr=new char[251];
  strcpy(virhe,"virhe: Liian monta nimi-arvo paria (max
35)");
  tulos=new char[10];
  strcpy(tulos,"null=null");

  }




void form::lue_stdin(){

        syote = NULL;


        syote = getenv("CONTENT_TYPE");
```

```
 //MIME-tyypin luku ympäristömuuttujasta

        if (syote == NULL || strcmp(syote,
"application/x-www-form-urlencoded"))
        {
                cout<<"<p>Otsikkokentän Content-Type:n
MIME-tyyppi puuttuu tai on sopimaton</body></html>";
                exit(1);
        }

        syote = getenv("CONTENT_LENGTH");
//Lähetetyn merkkijonon pituuden luku ympäristömuuttujas-
ta
        if (syote == NULL)
        {
                cout<<"<p>Content-type otsikkokentän pi-
tuus puuttuu! Katso CONTENT_LENGTH.</body></html>";
                exit(1);
        }

        koko = atoi(syote);
//Muutetaan luvuksi
        syote = new char[koko + 2];
//Varataan muistia

        if (syote == NULL)
        cout<<"<P>stdin puskuri on tyhjä!</BODY></HTML>";


        cin.getline(syote,koko+1);

        syote[koko+1]='\0';


}



void form::kasittele(){


lue_stdin();


pmaara=0;               // Parimäärä
nlaskin=0;              //Parien indeksi
```

```
n=0;                    //Nimikentän merkki-indeksi
m=0;            //Arvokentän merkki-indeksi
check_n=0;      //Tarkistus ylitetäänkö nimikentän max.
merkkimäärä
check_a=0;      //Tarkistus ylitetäänkö arvokentän max.
merkkimäärä
check_p=0;     //Tarkistus ylitetäänkö parimäärä
laskin=2;    // Muuttuja%2=0 tai !=0 -> parillinen tai
pariton


/* Nimi - arvo pareiksi hajottaminen */

for(int j=0;j<koko;j++)
                {

        apu_mr=syote[j];
 //Otetaan syötteen merkki käsittelyyn



          //Erotetaan nimikentän merkit

        if(apu_mr!='=' && apu_mr!='&' && laskin%2==0)
                        {       p[nlaskin].nimi[n]=apu_mr;
                                n++;
                                if(n>14)
                  //Nimikentän koon tarkistus

                                {       n--;
                                        check_n=1;
                                }
                        }


        //Erotetaan arvokentän merkit

        if(apu_mr!='&' && apu_mr!='=' && laskin%2!=0)
                        {       p[nlaskin].arvo[m]=apu_mr;
                                m++;
                                if(m>249)
                         //Arvokentän koon tarkistus
                                        {       m--;
                                                check_a=1;
                                        }
                        }
```

```
//Lisätään loppumerkki nimikenttään ja muutetaan laskin
parittomaksi

                        if(apu_mr=='=')
                        {

        if(check_n)p[nlaskin].nimi[n]='*';  //Lisätään
tähti loppuun jos koko ylitetty
                                p[nlaskin].nimi[n+1]='\0';
                                check_n=0;
//Nollataan koon tarkistus
                                laskin++;     //-> parit-
tomaksi
                                n=0;
                        //Nollataan nimikentän merkki-
                indeksi
                        }



         //Lisätään loppumerkki arvokenttään ja muutetaan
laskin parilliseksi

                        if(apu_mr=='&')
                        {

        if(check_a)p[nlaskin].arvo[m]='*';
//Lisätään tähti loppuun jos koko ylitetty
                                p[nlaskin].arvo[m+1]='\0';
                                nlaskin++;
//Lisätään parilaskuria yhdellä
                                check_a=0;
 //Nollataan koon tarkistus
                                if(nlaskin>49)
 //Parien lukumäärän tarkistus
                                        {       nlaskin--;
                                                check_p=1;
                                        }
                                laskin++;
 //-> parilliseksi
                                m=0;
 //Nollataan arvokentän merkki-indeksi
                        }

                }
```

```
 pmaara=nlaskin+1;

if(check_a)p[nlaskin].arvo[m]='*';
 //arvokentän  max. merkkimäärän ylitys -> viimeisek-
si(100) * -merkki
p[nlaskin].arvo[m+1]='\0';
 //Loppumerkin lisäys
if(check_p)strcpy(p[nlaskin].arvo,virhe);
 //Parimäärän ylitys -> arvokenttään virheilmoitus
if(check_p)strcpy(p[nlaskin].nimi,"virhe");
 //Parimäärän ylitys -> nimikenttään "virhe"



//Heksakoodattujen merkkien etsintä ja korjaus

/* ä=%E4 Ä=%C5 Ö=%D6 ö=%F6 å=%E5 Å=%C5 /=%2F (=%28 )=%29
#=%23
        ¤=%A4 [=%5B ]=%5D ~=%7E '=%27 §=%A7 ½=%BD ?=%3F
!=%21 ,=%2C
        ;=%3B :=%3A $=%24 |=%7C <=%3C >=%3E \=%5C {=%7B
}=%7D £=%A3  */

//ÄLÄ KÄYTÄ YLIMÄÄRÄISIÄ & TAI = MERKKEJÄ


 for(x=0;x<pmaara;x++)
        {

        //chm sisältää etsityn merkkijonon alkuosoitteen

                chm=strstr(p[x].arvo,"%E4");

                        while(chm!=NULL)
                             {

        strcpy(test_mr,chm+3);
//Otetaan talteen erikoismerkin jälkeinen osa
                                        strcpy(chm,"ä");
                //Sijoitetaan oikea merkki paikalleen

        strcat(p[x].arvo,test_mr);  //Yhdistetään alku-
ja loppuosa

        chm=strstr(chm,"%E4");
```

```
//Katsotaan onko lisää samoja erik. merkkejä

                                }

         }


 for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%C4");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
                                        strcpy(chm,"Ä");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%C4");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%D6");

                        while(chm!=NULL)
                                {
        strcpy(test_mr,chm+3);
        strcpy(chm,"Ö");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%D6");

                                }

         }
```

```
 for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%F6");

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+3);
        strcpy(chm,"ö");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%F6");

                                  }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%E5");

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+3);
        strcpy(chm,"å");

        strcat(p[x].arvo,test_mr);
        chm=strstr(chm,"%E5");
                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%C5");

                        while(chm!=NULL)
                              {
```

```
        strcpy(test_mr,chm+3);
        strcpy(chm,"Å");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%C5");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%2F");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"/");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%2F");

                                }

         }

for(x=0;x<pmaara;x++)
        {
        chm=strstr(p[x].arvo,"%28");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"(");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%28");

                                }
```

```
         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%29");

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+3);
        strcpy(chm,")");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%29");

                              }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%3F");

                        while(chm!=NULL)
                              {
        strcpy(test_mr,chm+3);
        strcpy(chm,"?");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%3F");

                              }

         }

for(x=0;x<pmaara;x++)
        {
```

```
                chm=strstr(p[x].arvo,"%3B");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,";");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%3B");

                                }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%3A");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,":");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%3A");
                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%24");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"$");
```

```
        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%24");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%7C");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"|");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%7C");

                                }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%3C");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"<");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%3C");

                                }
```

```
         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%3E");

                          while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,">");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%3E");

                                }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%7B");
          while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"{");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%7B");

                                }

         }


for(x=0;x<pmaara;x++)
        {
```

```
                chm=strstr(p[x].arvo,"%7D");

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+3);
        strcpy(chm,"}");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%7D");

                              }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%A3");

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+3);
        strcpy(chm,"£");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%A3");

                              }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%A4");

                        while(chm!=NULL)
                              {
```

```
        strcpy(test_mr,chm+3);
        strcpy(chm,"¤");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%A4");

                                }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%5B");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"[");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%5B");
                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%5D");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"]");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%5D");
```

```
                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%23");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"#");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%23");

                                }

         }

for(x=0;x<pmaara;x++)
        {
        chm=strstr(p[x].arvo,"%27");

        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"'");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%27");

                                }

         }


for(x=0;x<pmaara;x++)
        {
```

```
                chm=strstr(p[x].arvo,"%A7");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"§");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%A7");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%BD");

                        while(chm!=NULL)
                                {
        strcpy(test_mr,chm+3);
        strcpy(chm,"½");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%BD");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%21");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"!");
```

```
        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%21");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%7E");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"~");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%7E");

                                }

         }

for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%2C");

                        while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,",");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%2C");

                                }
```

```
         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%5C");

                          while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"\\");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%5C");

                                }

         }


for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%0D");
 //Vaunun palautus

                          while(chm!=NULL)
                                {

        strcpy(test_mr,chm+3);
        strcpy(chm,"");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%0D");

                                }

         }
```

```
for(x=0;x<pmaara;x++)
        {


                chm=strstr(p[x].arvo,"%0A");
 //Rivin vaihto

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+3);
        strcpy(chm,"<BR>");

        strcat(p[x].arvo,test_mr);

        chm=strstr(chm,"%0A");

                              }

         }


/*Välilyönnit tulevat syötteessä + merkkeinä -> muutetaan
takaisin välilyönneiksi */

for(x=0;x<pmaara;x++)
        {


                  chm=strchr(p[x].arvo,'+');

                        while(chm!=NULL)
                              {

        strcpy(test_mr,chm+1);
                strcpy(chm," ");

        strcat(p[x].arvo,test_mr);

        chm=strchr(chm+1,'+');
                              }

        }

}
```

```
void form::tallenna(){

ofstream kir-
jaa("data/arvoparit.dat",ios::trunc|ios::app);
/* Avataan tiedosto loppuun lisäämistä varten  ja tyhjen-
netään mahdollinen entinen */

delete tulos;

for(int z=0;z<pmaara;z++)
        {

                tulos=new
char[strlen(p[z].nimi)+strlen(p[z].arvo)+2];

        sprintf(tulos,"%s=%s",p[z].nimi,p[z].arvo);
                kirjaa<<tulos<<"\n\0";
                kirjaa.flush();
//Pakotetaan puskurin sisältö levylle
        }

system("chmod 755 data/arvoparit.dat");   //Oikeuksien
asetus

}
```

```
void form::print(){

cout<<"\nsyote = "<<syote<<"\n";

for(int k=0;k<nlaskin+1;k++)
        {
```

```
                cout<<"p["<<k<<"].nimi =
"<<p[k].nimi<<"\n";
                cout<<"p["<<k<<"].arvo =
"<<p[k].arvo<<"\n";

        }
}
```

```
void form::printhtml_script(){

cout<<"<BR><HR size='3' noshade><BR><font size='4' co-
lor='#ff0000'>";
cout<<"<BR>stdin puskurin sisältö on: "<<syote<<"<BR>";
cout<<"<BR> Nimi=arvo -parien lukumäärä =
"<<pmaara<<"<BR><BR>";
cout<<"Nimi=arvo -parit:<BR><BR>";

for(int k=0;k<pmaara;k++)
        {
                cout<<"p["<<k<<"].nimi =
"<<p[k].nimi<<"<BR>";
                cout<<"p["<<k<<"].arvo =
"<<p[k].arvo<<"<BR>";
                cout<<"<BR>";
        }

cout<<"</BODY></HTML>";

}
```

```
void main(int argc, char *argv[]){

form f;
char *komento,*param_1,*param_2;
int ei_ole=0;

if(argc>1)param_1=new char[strlen(argv[1])];
 //Varataan muistia parametreille
if(argc>2)param_2=new char[strlen(argv[2])];

if(argc==1)                        //Jos ei parametreja
        {
        param_1=new char[11];
        strcpy(param_1,"cgi_script");
        }


if(argc>1)strcpy(param_1,argv[1]);
//Jos yksi parametri tai enemmän
if(argc>2)strcpy(param_2,argv[2]);
 //Jos kaksi parametria tai enemmän

if(argc>3)
//Jos ylimääräisiä parametreja
        {
        cout<<"Content-type: text/html\n";
        cout<<"\n";
        cout<<"<HTML><BODY bgcolor='#ffffff'><font si-
ze='5' color='#ff0000'><CENTER>";
        cout<<"<P>Parametrien lukumäärä suurempi kuin
kaksi(2)!<BR><BR>";
        cout<<"Tarkista URL.</CENTER></BODY></HTML>";
   exit(1);
   }


f.kasittele();
 //Puretaan nimi=arvo -parit

f.tallenna();


fstream koe(param_1,ios::out|ios::noreplace);
 //Avaus epäonnistuu jos tiedosto on olemassa

         if(koe.good())
```

```
         {
        cout<<"Content-type: text/html\n";
        cout<<"\n";
        cout<<"<HTML><BODY bgcolor='#ffffff'><font si-
ze='4' color='#ff0000'><B>\n";
         cout<<"<BR><CENTER>Tiedostoa "<<param_1<<" ei
löytynyt!\n";

        cout<<"</B></CENTER></BODY></HTML>";
         koe.close();
         komento=new char[strlen(param_1)+4];
         sprintf(komento,"rm %s",param_1);
        //Poistetaan ylimääräinen tiedosto
        system(komento);
         delete komento;
         ei_ole=1;
         }



if(!ei_ole)       //Suoritetaan jos tiedosto olemassa
        {
        komento=new char[strlen(param_1)+3];
//Varataan muistia

        sprintf(komento,"./%s",param_1);
        system(komento);
  //Ajetaan parametrina välitetty scripti
        delete komento;
   }

if(argc>2 &&
!strcmp(param_2,"DEBUG"))f.printhtml_script();
//Jos param_2:n arvo DEBUG


delete param_1;
if(argc>2)delete param_2;

}
```

Ohjelma löytyy kirjan mukana seuraavalta levyltä nimellä **form.cpp**. Se käännetään unix-ympäristöön (testattu **gnu**-kääntäjällä) komennolla

```
g++ form.cpp
```

Kääntäjä tekee tiedoston **a.out**. Muutetaan käännetty ohjelma **form**-nimiseksi komennolla ja siirretään se palvelimen ohjelmahakemistoon (cgi-bin).

```
mv a.out form
```

Seuraavassa **arvoparit.dat** -tiedostoa lukevan ohjelman (**get_f**) koodi (C -kielinen ohjelma). Sille annetaan parametrinä mikä rivi luetaan ja luetaanko nimi- vai arvokenttä. Tarkempi selostus ohjelmakoodin alussa. Ohjelma löytyy kirjan mukana seuraavalta levyltä nimellä **get_f.c**. Ohjelma käännetään unix-ympäristöön (testattu **gnu**-kääntäjällä) seuraavalla komennolla.

```
gcc get_f.c
```

Kääntäjä tekee tiedoston **a.out**. Muutetaan käännetty ohjelma get_f-nimiseksi komennolla ja siirretään se palvelimen ohjelmahakemistoon (cgi-bin).

```
mv a.out get_f
```

Lähdekoodi: get_f.c

```
/* WWW-lomakkeen käsittelyohjelman (form.cpp) arvoparien
lukuohjelma  versio 1.1
tiedostosta data/arvoparit.dat Juha Niemi 7.9.1996  */

/* Arvoparit on kirjoitettu (form-ohjelmalla) tiedostoon
riveittäin muodossa nimi=arvo siten, että ensimmäinen lo-
makkeen nimi=arvo -pari on ensimmäisellä rivillä
```

```
(rivinumero 0) ja toinen  lomakkeen nimi=arvo -pari on
toisella rivillä (rivinumero 1) jne.

*/


/*Parametrin kautta välitetään tieto luetaanko scriptiin
nimikenttä (nx) vai arvokenttä ax. x:n paikalle sijoite-
taan rivinumero, HUOM! alkaen nollasta. */

/* Esimerkki scripti  tulostaa kolmannen rivin nimi=arvo
parin lukuohjelmaan (WWW)
jotkut järjestelmät vaativat kaksoispisteen (:) komento-
tiedoston alkuun varsinkinjos se ajetaan komentoriviltä
(testaus).
CGI-ohjelmana ajettaessa siitä saattaa tulla (ja yleensä
tuleekin) server-errorvirheilmoitus.
Jos scriptit on kirjoitettu DOS-koneessa, saattavat ^M -
merkit (rivinvaihto, näkyvät vi:llä)aiheuttaa ongelmia ja
ne kannattaa pyyhiä pois ennen ajamista unix-
koneessa(Aiheuttavat mahdollisen server error ilmoituk-
sen).
echo "Con.. ja echo "" vaaditaan kun tulostetaan www-
lukuohjelmaan


:
#!/bin/sh
echo "Content-type: text/htnl"
echo ""
muuttuja1=`get_f n2`
muuttuja2=`get_f a2`
echo $muuttuja1
echo $muuttuja2

*/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

FILE *f;


void main(int argc, char *argv[]){
```

```
int valitsin=0,rivi=0,r_laskin=0,pituus=0,index=0,i=0;
char ni-
mi[16]="null",arvo[101]="null",param[31],*chm,*check,db_r
ivi[255],
apu_mr[2];



if(argc>1)strcpy(param,argv[1]);

if(argc==1)strcpy(param,"n0");
 /* oletusarvo jos parametria ei annettu */

if(param[0]=='n')valitsin=1;
/* tarkistetaan halutaanko nimi- vai arvokenttä */

if(param[0]=='a')valitsin=2;

apu_mr[0]=param[1];      /* Otetaan talteen parametrin
lukuosa */
apu_mr[1]='\0';

rivi=atoi(apu_mr);      /* muutetaan char-tyyppi luvuksi
*/

f=fopen("data/arvoparit.dat","r");

        if (f==NULL)
          {
                  printf("error: tiedostoa da-
ta/arvoparit.dat");
                  printf(" ei löytynyt. Ohjelman ajo lope-
tettu !");
                  exit(1);
          }

/*luetaan tietokannan rivi ja pakotetaan puskuri tyhjäk-
si*/

check=fgets(db_rivi,sizeof(db_rivi),f);
fflush(f);

while(check!=NULL)

        {
```

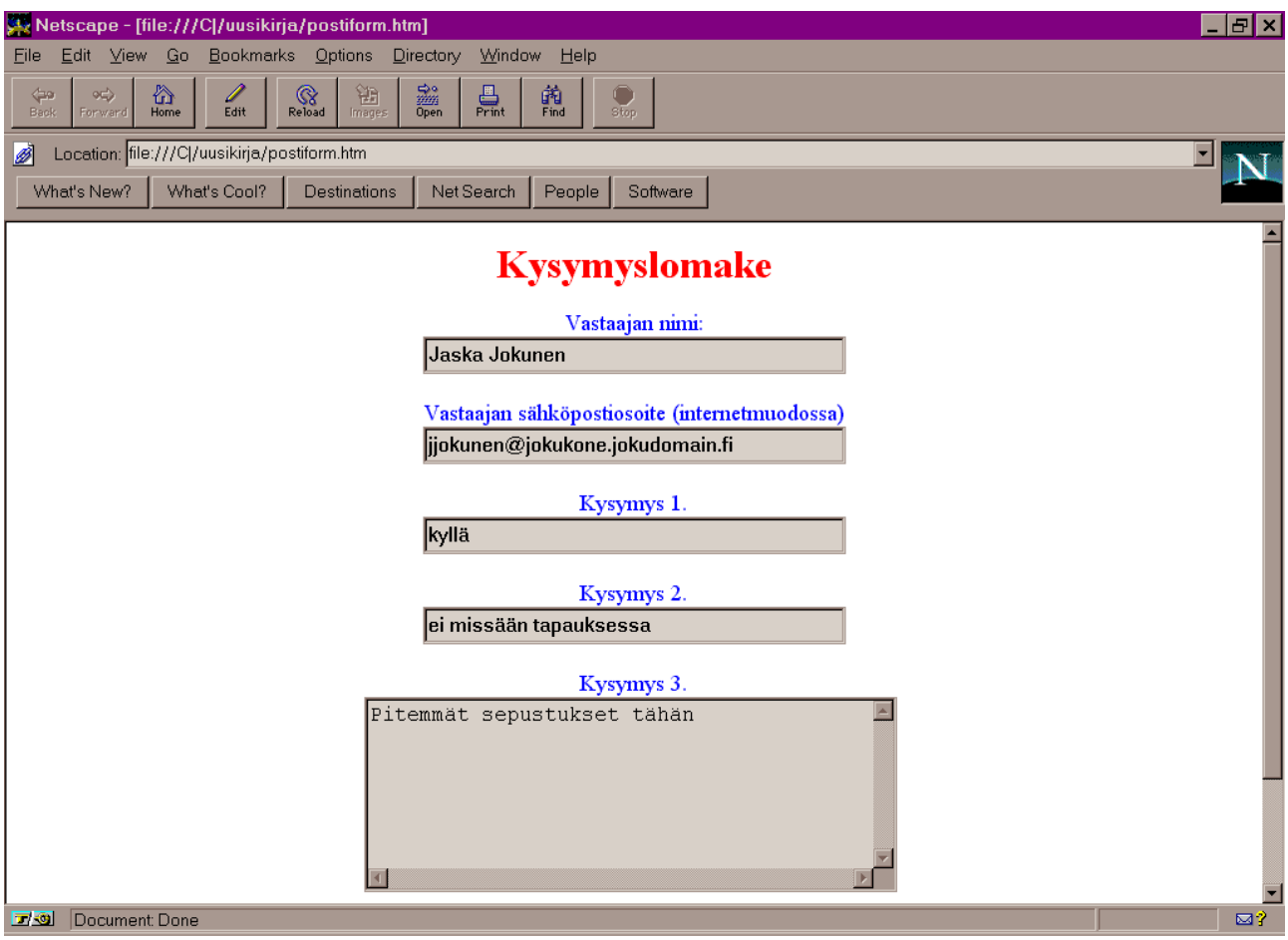
```
        if(r_laskin==rivi)
 /* Jos oikea rivi -> suoritetaan */
        {
                pituus=strlen(db_rivi);
                for(i=0;i<pituus;i++)
                {
                if(db_rivi[i]=='=')index=i;
/* merkitään = -merkin paikka (indeksi) */
                }


                if(valitsin==1 )
                        {
                        strncpy(nimi,db_rivi,index);
  /* luetaan nimikenttä ja lisätään loppumerkki */

                        nimi[index]='\0';
                        }

                if(valitsin==2)
                        {
                        chm=strchr(db_rivi,'=');
/* katsotaan = -merkin osoite */

                        strcpy(arvo,chm+1);
 /* luetaan arvokenttä ja lisätään loppumerkki rivinvaih-
tomerkin paikalle*/
                        chm=strchr(arvo,'\n');
                                strcpy(chm,"\0");
                         }

         }

        r_laskin++;           /* rivilaskuri */

        check=fgets(db_rivi,sizeof(db_rivi),f);
 /* luetaan uusi rivi ja pakotetaan puskuri tyhjäksi */
        fflush(f);


        }

        if(valitsin==1)printf("%s",nimi);
/* tulostetaan parametrilla valittu kenttä */
        if(valitsin==2)printf("%s",arvo);
}
```

Seuraavassa Esimerkki ohjelmien käytöstä. Kyselylomake lähetetään POST-lähetystavalla, se käsitellään ensin **form** -ohjelmalla ja sitten ajetaan **posti**-niminen komentotiedosto, joka kirjaa tiedostoon **postilista** kaikki lähetetyt sähköpostit ja tekee tiedoston **lahetys**, joka lähetetään vastaanottajalle ja pyyhitään pois sen jälkeen. Lisäksi ajetaan DEBUG -toiminto.

Kyselylomakesivu.

```
<HTML>
<BODY bgcolor="#ffffff">

<CENTER></B><font color="#ff0000">
<H1>Kysymyslomake</H1>
</B></CENTER>

<FORM  method="post"
ACTION="http://impedanssi.it.spt.fi:8998/cgi-
bin/form?posti+DEBUG">
<INPUT TYPE="hidden" NAME="osoite"
VALUE="emijuha@jazz.cc.spt.fi">

<font color="#0000ff" size="4">

<CENTER>
Vastaajan nimi:<BR>
<INPUT TYPE="text" NAME="eka" SIZE="40"><BR><BR>

Vastaajan sähköpostiosoite (internetmuodossa)<BR>
<INPUT TYPE="text" NAME="toka" SIZE="40"
VALUE="tunnus@kone.domain.fi"><BR><BR>

Kysymys 1.<BR>
<INPUT TYPE="text" NAME="kolmas" SIZE="40"><BR><BR>

Kysymys 2.<BR>
<INPUT TYPE="text" NAME="neljas" SIZE="40"><BR><BR>
```

```
Kysymys 3.<BR>
<TEXTAREA NAME="teksti" ROWS="6" COLS="40">
Pitemmät sepustukset tähän
</TEXTAREA>
<BR><BR>
<TABLE border="4"><TR><TD>
<TABLE border="2"><TR><TD>
<INPUT TYPE="submit" value="Lähetä vastaus">
</TD></TR></TABLE></TD></TR></TABLE>
</CENTER></FORM>
</BODY>
</HTML>
```

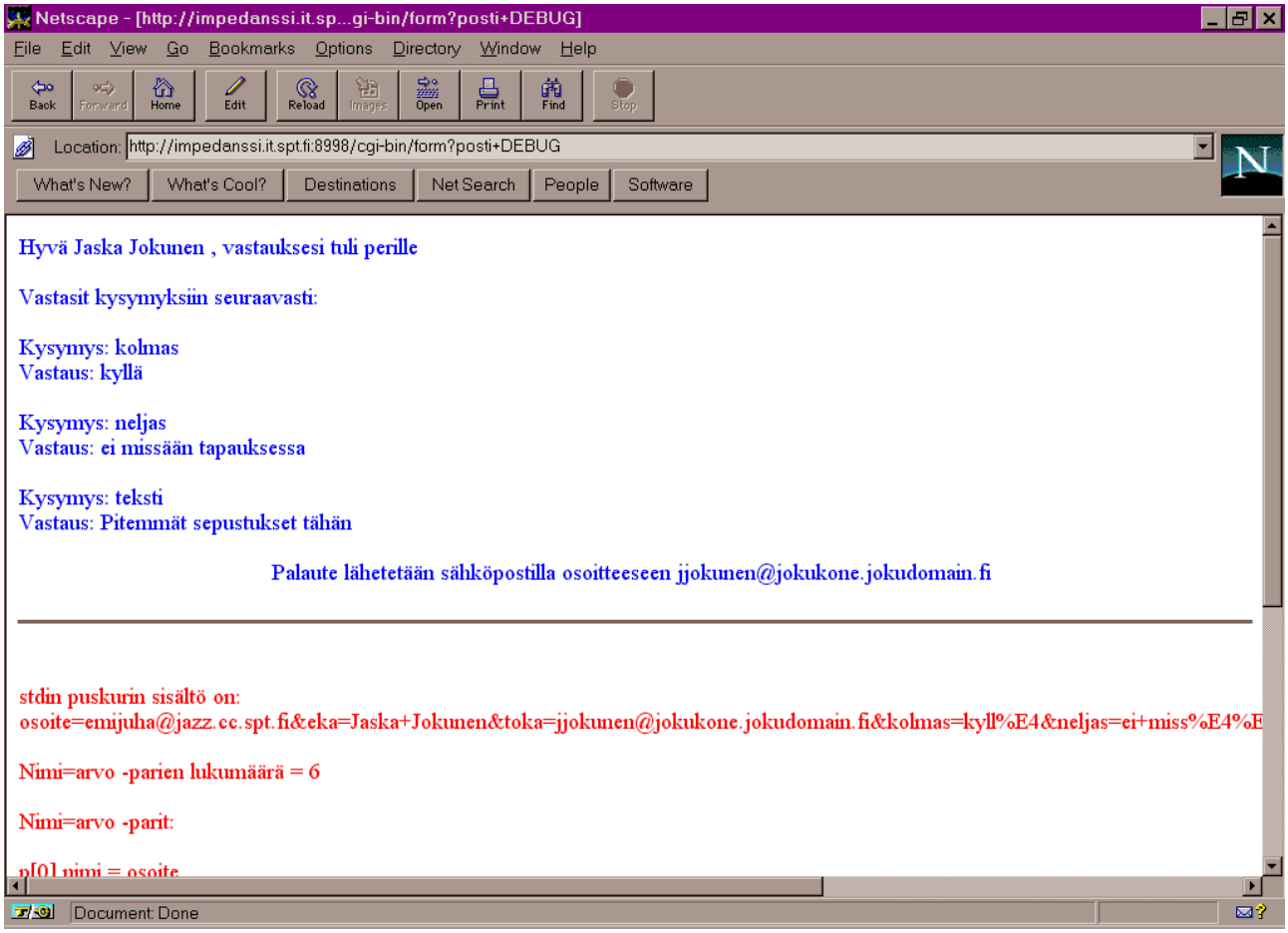


Edellinen kysymyslomake kun kenttiin on kirjoitettu vastaukset.

Komentotiedosto posti.

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
n0=`./get_f n0`
a0=`./get_f a0`
n1=`./get_f n1`
a1=`./get_f a1`
n2=`./get_f n2`
a2=`./get_f a2`
n3=`./get_f n3`
a3=`./get_f a3`
n4=`./get_f n4`
a4=`./get_f a4`
n5=`./get_f n5`
a5=`./get_f a5`
pva=`date`
cat << EndOfHeader| cat >> postilista
Vastaajan $a1
Mail $a2
Päiväys: $pva
Vastaukset olivat seuraavat:

$n3 = $a3
$n4 = $a4
$n5 = $a5
---------------------------------------
EndOfHeader
cat << EndOfHeader > lahetys
Vastaajan $a1
Mail $a2
Päiväys: $pva
Vastaukset olivat seuraavat:

$n3 = $a3
$n4 = $a4
$n5 = $a5
---------------------------------------
EndOfHeader
cat lahetys|mail $a0
cat << EndOfHeader
<HTML>
<BODY bgcolor='#ffffff'>
<font size='4' color='#0000ff'>
<P>
Hyvä $a1 , vastauksesi tuli perille
```

```
<P>
Vastasit kysymyksiin seuraavasti:
<P>
Kysymys: $n3 <BR>
Vastaus: $a3 <BR><BR>

Kysymys: $n4 <BR>
Vastaus: $a4 <BR><BR>

Kysymys: $n5 <BR>
Vastaus: $a5

<P>
<CENTER>
Palaute lähetetään sähköpostilla osoitteeseen $a2

</CENTER>
</BODY>
</HTML>
EndOfHeader
rm lahetys
```



Posti -komentotiedoston aikaansaama tuloste kun lomake lähetettiin.

Posti -komentotiedoston lähettämä sähköposti.

Return-Path: <jniemi@it.spt.fi>
Date: Mon, 23 Sep 1996 18:19:49 -0200
From: jniemi@impedanssi.it.spt.fi (Juha Niemi)
To: emijuha@jazz.cc.spt.fi

Vastaajan Jaska Jokunen
Mail jjokunen@jokukone.jokudomain.fi
Päiväys: Mon Sep 23 18:19:48 GMT 1996
Vastaukset olivat seuraavat:

kolmas = kyllä
neljas = ei missään tapauksessa
teksti = Pitemmät sepustukset tähän

HAKEMISTO

A

absbottom 38, 39
absmiddle 38, 39
action-funktio 149
align 23, 24, 33, 38, 39, 95, 113, 121, 152
alt 5, 33, 38, 39, 44, 67, 69, 72, 82, 87, 127, 136, 159, 183

B

bash 223
bgcolor 20, 21, 39, 68, 121
bin/sh 225
border 33, 34, 38, 39, 47, 62, 95, 113, 121, 151, 152, 169, 171
Break-lause 83
Button-luokka 154

C

Cache 11
case-lause 236
cat 230
CGI
ohjelma 5, 7, 51, 52, 65, 184
cgi-bin 221
Checkbox-luokka 157
chmod 223
color 20, 21, 22, 23, 39
Common Gateway Interface 221
content 21, 38
Content-type 148, 222, 223, 225, 232, 241, 243, 245, 256, 257, 259, 280, 283, 287
Continue-lause 83
Cookie 218
Ctrl-M 223

D

Date-luokka 136
Document-olio 149, 183, 193
Do-while-lause 83

E

echo 225
Elementit 149, 154
ENCTYPE 150
exit 233

F

FILE 49
For-lause 82
form.cpp 254, 281, 282
Form-olio 147, 149, 154, 183
Frame-olio 178
FTP 47

G

***gravis*-merkki** 229

H

height 39, 198
History-olio 190
hspace 38, 39
HTML-kieli
komento
tagi
HTTP 47
http-equiv 22, 38

I

If-else-lause 81
Imagemap- 44
index.html 37
Interaktiiviset kuvat 44

J

Java 9

K

Karttatiedosto 46
komentotulkki 223
ksh 223
Kuuma kohta 45

L

Link-olio 159
Location-olio 192
ls 224

M

Magneettinen piste 46
MAILTO 49
Math-luokka 76, 110
MIME-tyyppi 186, 187
mkdir 233

N

Navigaattori-olio 183

Navigator-olio 181
New-operaattori 102
NEWS 48
nimi=arvo 238, 240, 244, 248, 254, 255, 256, 280, 282

O

Oikeuksien tunnukset 224
otsikkotieto 19, 21, 22

P

Password-olio 160
protokolla 36, 37, 47, 48, 49, 193, 216, 217
public_html 222
Putkimekanismi 232
päivitystaajuus 38

R

RadioButton-olio 161
Reset-olio 165

S

SCRIPT-tagi 99
Selection-olio 167
Set-Cookie 217
sh 223
Solaris 221
String-luokka 75
Submit-olio 173
Syötteen ohjaus 231

T

tag 19
Textarea-olio 178
Text-olio 174
This-muuttuja 92
tiedoston oikeudet 224

U

Un-CGI 253

W

WAIS 49
Varatut sanat 71
Var-lause 71
While-lause 83
vi 223
width 33, 39, 198
Window-olio 193
With-lause 84
vspace 38